

UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

**Algoritmo Genético para Geração
de Ondas de Inimigos em Jogos**

Daniel Yoshio Hotta
Rafael Gonçalves Pereira da Silva
Ricardo Akira Tanaka

MONOGRAFIA FINAL

MAC 499 — TRABALHO DE
FORMATURA SUPERVISIONADO

Supervisor: Prof. Dr. Marco Dimas Gubitoso
Cossupervisor: Wilson Kazuo Mizutani

São Paulo
24 de Dezembro de 2021

Resumo

Daniel Yoshio Hotta Rafael Gonçalves Pereira da Silva Ricardo Akira Tanaka. **Algoritmo Genético para Geração de Ondas de Inimigos em Jogos**. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2021.

Este trabalho se propõe a estudar a viabilidade de um algoritmo evolutivo para geração de ondas de oponentes em jogos cujo o objetivo do jogador é eliminá-los. Para ampliar o escopo do projeto, foi definido que o algoritmo desenvolvido deveria ser o mais genérico possível, possibilitando sua implementação em diversos jogos, apenas com pequenos refatoramentos do código. Considerando a existência de ondas de inimigos, cujos individuais podem obter sucesso ou fracasso contra o jogador, definiu-se que um algoritmo genético seria apropriado, já que o mesmo poderia usar as ondas como população e conseguiria obter candidatos de sucesso para a próxima onda, sem a necessidade de características específicas do jogo, e se adaptando a mudanças nas condições ou da estratégia de quem está jogando. Foram desenvolvidos dois protótipos de jogos dentro das categorias delimitadas, um *Tower Defense* e um *Top-Down Shooter*, com ferramentas de código aberto como a *game engine Godot*. Os jogos foram selecionados por possuírem suficientes características comuns entre si, permitindo a implementação do mesmo algoritmo com pequenas adaptações, e ao mesmo tempo diferentes o suficiente para produzirem experiências e resultados diferentes. Durante o desenvolvimento diversos fatores e decisões se mostraram essenciais ao algoritmo, como diferentes necessidades de *fitness* e mutações em cada jogo e entre ondas de inimigos. Finalmente foram desenvolvidos métodos de teste e métricas de avaliação para, com dados objetivos, obter indícios de que o algoritmo seria superior a implementações ingênuas como uma geração aleatória. Os resultados aparentam maior adequação a jogos mais determinísticos, onde um ambiente mais estático acaba favorecendo o algoritmo.

Palavras-chave: algoritmos genéticos. algoritmos evolutivos. geração de inimigos. jogos.

Abstract

Daniel Yoshio Hotta Rafael Gonçalves Pereira da Silva Ricardo Akira Tanaka. **Genetic Algorithm for Generation of Enemy Waves in Games**. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2021.

This project is a study around the viability of an evolutionary algorithm for generating waves of enemies in games, where the player's objective is eliminating them. To expand the subject, it was defined that the algorithm should be as generic as possible to allow its implementation in other games, with minor code refactoring. Considering the enemy wave as a group of individuals that can end up in a success or failure in the round, a genetic algorithm seemed suitable, since it would be able to use the waves as population, with each individual a candidate for the next wave, representing the next generation of individuals. The algorithm does not depend on extensive knowledge about the game mechanics, being able to adapt to ambient changes and player strategy. Two game prototypes were developed, a Tower Defense and a Top-Down Shooter, using open source tools such as the Godot Game Engine. The games were selected due to its common characteristics in style, which allows using the same evolutionary algorithm, but at the same time they diverge in gameplay and strategy, producing different results and experiences. During development several factors and decisions were key to the algorithm, such as the fitness and mutation calculations for each game, and each wave. To enable performance analysis, testing methods and evaluation metrics were developed, enabling objective data to be gathered and measured against randomly generating enemies. Data analysis showed that the genetic algorithm performs above other naive methods, but is more successful in deterministic games, where a more stable environment favors the algorithm.

Keywords: genetic algorithms. evolutionary algorithm. enemy generator. games.

Lista de Abreviaturas

ABNT	Associação Brasileira de Normas Técnicas
URL	Localizador Uniforme de Recursos (<i>Uniform Resource Locator</i>)
IME	Instituto de Matemática e Estatística
USP	Universidade de São Paulo
TD	Tower Defense
SS	Space Shooter
IA	Inteligência Artificial

Lista de Figuras

2.1	Kingdom Rush. Obtida de https://www.ironhidegames.com/Games/kingdom-rush	4
2.2	Space Invaders. Obtida de https://www.gamekult.com/jeux/space-invaders-17318/test.html	5
2.3	Asteroids. Obtida de https://openclipart.org/detail/281726/asteroids-video-games-1979	5
3.1	Godot Game Engine.	7
3.2	Árvore no Godot.	8
3.3	Utilizando nós CollisionShape2D e Area2D. Imagem obtida do editor de jogo.	8
3.4	Árvore formada por nós no Godot. Em Player um ícone indica a presença de um script vinculado ao nó. Imagem obtida da documentação em https://docs.godotengine.org/en/stable/getting_started/step_by_step/scenes_and_nodes.html	9
3.5	Instanciação de tiros no Space Shooter. Imagem retirada do próprio jogo.	10
4.1	Tela Inicial. Imagem retirada do próprio jogo.	14
4.2	Área de detecção torre verde. Imagem retirada do próprio jogo.	14
4.3	Área de detecção torre vermelha. Imagem retirada do próprio jogo.	15
4.4	Exemplo de uma execução do jogo. Imagem retirada do próprio jogo.	15
4.5	Tipos de tanques	16
4.6	Definição das rotas Norte e Sul. Imagem retirada do jogo e editada.	17
4.7	Tela inicial. Imagem retirada do próprio jogo.	18
4.8	Disparos	18
4.9	Opções de Player Automático. Imagem retirada do próprio jogo.	18
4.10	Tipos de asteroides	19
4.11	Exemplo de uma rodada do jogo. Imagem retirada do próprio jogo.	20
4.12	Posições possíveis onde um asteroide pode surgir. Imagem retirada do jogo e editada.	20

5.1	Estrutura de um algoritmo genético.	22
5.2	<i>Crossover</i> para o caso do TD.	24
6.1	Tela de Teste Tower defense. Imagem retirada do próprio jogo.	29
6.2	Tela de Teste Space Shooter. Imagem retirada do próprio jogo.	29
6.3	Menu de testes Tower Defense. Imagem retirada do próprio jogo.	30
6.4	Menu de Testes Space Shooter. Imagem retirada do próprio jogo.	30
6.5	Escolha Random em tower defense. Imagem retirada do próprio jogo.	31
6.6	Escolha One Each no Tower Defense. Imagem retirada do próprio jogo.	31
6.7	Opção All Orange. Imagem retirada do próprio jogo.	32
6.8	Teste de torres verdes	33
6.9	Teste de torres vermelhas	33
6.10	Teste de torres verdes com vermelhas	34
6.11	Teste de torres vermelhas com verdes	34
6.12	Área de detecção da nave em um modo de teste. Imagem retirada do próprio jogo.	35
6.13	Movimentação da nave em um modo de teste. Imagem retirada do jogo e editada.	35
7.1	Visualização das ondas aleatórias com maior dano no Tower Defense.	41
7.2	Visualização das ondas aleatórias com maior dano no Space Shooter.	44
7.3	Gráfico com as médias de dano para cada onda no teste com as Torres Verdes para as versões v1, v2 e v3.	48
7.4	Gráfico com as médias de dano para cada onda no teste com as Torres Vermelhas para as versões v1, v2 e v3.	49
7.5	Gráfico com as médias de dano para cada onda no teste com as Torres Verdes + Vermelhas para as versões v1, v2 e v3.	49
7.6	Gráfico com as médias de dano para cada onda no teste com as Torres Vermelhas + Verdes para as versões v1, v2 e v3.	50
7.7	Boxplot do dano das 30 ondas nos 30 experimentos, para as versões v1, v2 e v3.	51
7.8	Boxplot do dano das 30 ondas nos 30 experimentos, para as versões v1, v2 e v3.	52
7.9	Boxplot do dano das 30 ondas nos 30 experimentos, para as versões v1, v2 e v3.	53
7.10	Boxplot do dano das 30 ondas nos 30 experimentos, para as versões v1, v2 e v3.	54

7.11	Gráfico com as médias de dano para cada onda no teste com a Nave Parada, Disparo Amarelo para as versões v1 e v3.	57
7.12	Gráfico com as médias de dano para cada onda no teste com a Nave Movendo, Disparo Amarelo para as versões v1 e v3.	57
7.13	Gráfico com as médias de dano para cada onda no teste com a Nave Parada, Disparo Vermelho para as versões v1 e v3.	58
7.14	Gráfico com as médias de dano para cada onda no teste com a Nave Movendo, Disparo Vermelho para as versões v1 e v3.	58
7.15	Boxplot do dano das 30 ondas nos 30 experimentos, para as versões v1 e v3.	59
7.16	Boxplot do dano das 30 ondas nos 30 experimentos, para as versões v1 e v3.	60
7.17	Boxplot do dano das 30 ondas nos 30 experimentos, para as versões v1 e v3.	61
7.18	Boxplot do dano das 30 ondas nos 30 experimentos, para as versões v1 e v3.	62
A.1	Visualização da moda de cada onda com a versão v1 contra Torres Verdes.	73
A.2	Visualização da moda de cada onda com a versão v1 contra Torres Verdes.	74
A.3	Visualização da moda de cada onda com a versão v1 contra Torres Verdes.	75
A.4	Visualização da moda de cada onda com a versão v1 contra Torres Vermelhas.	77
A.5	Visualização da moda de cada onda com a versão v1 contra Torres Vermelhas.	78
A.6	Visualização da moda de cada onda com a versão v1 contra Torres Vermelhas.	79
A.7	Visualização da moda de cada onda com a versão v1 contra Torres Verdes + Vermelhas.	81
A.8	Visualização da moda de cada onda com a versão v1 contra Torres Verdes + Vermelhas.	82
A.9	Visualização da moda de cada onda com a versão v1 contra Torres Verdes + Vermelhas.	83
A.10	Visualização da moda de cada onda com a versão v1 contra Torres Vermelhas + Verdes.	85
A.11	Visualização da moda de cada onda com a versão v1 contra Torres Vermelhas + Verdes.	86
A.12	Visualização da moda de cada onda com a versão v1 contra Torres Vermelhas + Verdes.	87
B.1	Visualização da moda de cada onda com a versão v2 contra Torres Verdes.	91
B.2	Visualização da moda de cada onda com a versão v2 contra Torres Verdes.	92
B.3	Visualização da moda de cada onda com a versão v2 contra Torres Verdes.	93
B.4	Visualização da moda de cada onda com a versão v2 contra Torres Vermelhas.	95
B.5	Visualização da moda de cada onda com a versão v2 contra Torres Vermelhas.	96
B.6	Visualização da moda de cada onda com a versão v2 contra Torres Vermelhas.	97

B.7	Visualização da moda de cada onda com a versão v2 contra Torres Verdes + Vermelhas.	99
B.8	Visualização da moda de cada onda com a versão v2 contra Torres Verdes + Vermelhas.	100
B.9	Visualização da moda de cada onda com a versão v2 contra Torres Verdes + Vermelhas.	101
B.10	Visualização da moda de cada onda com a versão v2 contra Torres Vermelhas + Verdes.	103
B.11	Visualização da moda de cada onda com a versão v2 contra Torres Vermelhas + Verdes.	104
B.12	Visualização da moda de cada onda com a versão v2 contra Torres Vermelhas + Verdes.	105
C.1	Visualização da moda de cada onda com a versão v3 contra Torres Verdes.	109
C.2	Visualização da moda de cada onda com a versão v3 contra Torres Verdes.	110
C.3	Visualização da moda de cada onda com a versão v3 contra Torres Verdes.	111
C.4	Visualização da moda de cada onda com a versão v3 contra Torres Vermelhas.	113
C.5	Visualização da moda de cada onda com a versão v3 contra Torres Vermelhas.	114
C.6	Visualização da moda de cada onda com a versão v3 contra Torres Vermelhas.	115
C.7	Visualização da moda de cada onda com a versão v3 contra Torres Verdes + Vermelhas.	117
C.8	Visualização da moda de cada onda com a versão v3 contra Torres Verdes + Vermelhas.	118
C.9	Visualização da moda de cada onda com a versão v3 contra Torres Verdes + Vermelhas.	119
C.10	Visualização da moda de cada onda com a versão v3 contra Torres Vermelhas + Verdes.	121
C.11	Visualização da moda de cada onda com a versão v3 contra Torres Vermelhas + Verdes.	122
C.12	Visualização da moda de cada onda com a versão v3 contra Torres Vermelhas + Verdes.	123
D.1	Visualização da moda de cada onda com a versão v1 contra Nave Parada, Disparo Amarelo.	126
D.2	Visualização da moda de cada onda com a versão v1 contra Nave Parada, Disparo Amarelo.	127
D.3	Visualização da moda de cada onda com a versão v1 contra Nave Parada, Disparo Amarelo.	128

D.4	Visualização da moda de cada onda com a versão v1 contra Nave Movendo, Disparo Amarelo.	129
D.5	Visualização da moda de cada onda com a versão v1 contra Nave Movendo, Disparo Amarelo.	130
D.6	Visualização da moda de cada onda com a versão v1 contra Nave Movendo, Disparo Amarelo.	131
D.7	Visualização da moda de cada onda com a versão v1 contra Nave Parada, Disparo Vermelho.	132
D.8	Visualização da moda de cada onda com a versão v1 contra Nave Parada, Disparo Vermelho.	133
D.9	Visualização da moda de cada onda com a versão v1 contra Nave Parada, Disparo Vermelho.	134
D.10	Visualização da moda de cada onda com a versão v1 contra Nave Movendo, Disparo Vermelho.	135
D.11	Visualização da moda de cada onda com a versão v1 contra Nave Movendo, Disparo Vermelho.	136
D.12	Visualização da moda de cada onda com a versão v1 contra Nave Movendo, Disparo Vermelho.	137
E.1	Visualização da moda de cada onda com o fitness v3 contra Nave Parada, Disparo Amarelo.	140
E.2	Visualização da moda de cada onda com o fitness v3 contra Nave Parada, Disparo Amarelo.	141
E.3	Visualização da moda de cada onda com o fitness v3 contra Nave Parada, Disparo Amarelo.	142
E.4	Visualização da moda de cada onda com o fitness v3 contra Nave Movendo, Disparo Amarelo.	143
E.5	Visualização da moda de cada onda com o fitness v3 contra Nave Movendo, Disparo Amarelo.	144
E.6	Visualização da moda de cada onda com o fitness v3 contra Nave Movendo, Disparo Amarelo.	145
E.7	Visualização da moda de cada onda com o fitness v3 contra Nave Parada, Disparo Vermelho.	146
E.8	Visualização da moda de cada onda com o fitness v3 contra Nave Parada, Disparo Vermelho.	147
E.9	Visualização da moda de cada onda com o fitness v3 contra Nave Parada, Disparo Vermelho.	148

E.10	Visualização da moda de cada onda com o fitness v3 contra Nave Movendo, Disparo Vermelho.	149
E.11	Visualização da moda de cada onda com o fitness v3 contra Nave Movendo, Disparo Vermelho.	150
E.12	Visualização da moda de cada onda com o fitness v3 contra Nave Movendo, Disparo Vermelho.	151

Lista de Tabelas

4.1	Dano das torres no Tower Defense.	15
4.2	Dano de cada tanque no Tower Defense	16
4.3	Dano dos tiros no Space Shooter	17
4.4	Velocidade, dano e vida de cada asteroide no Space Shooter	19
6.1	Exemplo de quadrado latino	27
6.2	Dano das torres no Tower Defense.	32
6.3	Dano de cada tanque no Tower Defense	32
7.1	Forma de coleta dos dados de cada onda.	37
7.2	Dano de cada tanque no Tower Defense	38
7.3	Desempenho das ondas repetidas contra Torres exclusivamente Verdes	38
7.4	Desempenho das ondas repetidas contra Torres exclusivamente Vermelhas	39
7.5	Desempenho das ondas repetidas contra 1 Torre Verde e 1 Vermelha (Nessa ordem)	39
7.6	Desempenho das ondas repetidas contra 1 Torre Vermelha e 1 Verde (Nessa ordem)	40
7.7	Média e Desvio Padrão do dano em ondas aleatórias no Tower Defense.	40
7.8	Ondas aleatórias com maior dano no Tower Defense.	41
7.9	Velocidade, dano e vida de cada asteroide no Space Shooter	41
7.10	Ondas repetidas contra IA de Disparo Amarelo e Parado	42
7.11	Ondas repetidas contra IA de Disparo Amarelo e Movendo	42
7.12	Ondas repetidas contra IA de Disparo Vermelho e Parado	43

7.13	Ondas repetidas contra IA de Disparo Vermelho e Movendo	43
7.14	Média e Desvio Padrão do dano em <i>ondas aleatórias</i> no Space Shooter. . .	44
7.15	Ondas aleatórias com maior dano no Space Shooter.	44
7.16	Relação dos fitness e taxa de mutação usados no TD e SS.	45
7.17	Tabela mostrando a onda onde ocorreu a convergência e o dano médio a partir desse ponto até o final.	55
7.18	Tabela mostrando a onda onde ocorreu a convergência e o dano médio a partir desse ponto até o final.	63
8.1	Dados agregados das médias de todos os testes, ordenado do maior para o menor no Tower Defense	66
8.2	Dados agregados das médias de todos os testes, ordenado do maior para o menor no Tower Defense	66
8.3	Dados agregados das médias de todos os testes, ordenado do maior para o menor no Space Shooter	67
8.4	Dados agregados das médias de todos os testes, ordenado do maior para o menor no Space Shooter	67

Lista de Programas

4.1	Sistema de Resistência TD.	16
5.1	Inicialização do <i>Space Shooter</i> com tipo de inimigo e local de surgimento.	22
5.2	Inicialização do <i>Tower Defense</i> com tipo de inimigo e rota.	22
5.3	<i>Fitness</i> do <i>Tower Defense</i> (Avaliação, em português).	23
5.4	<i>Fitness</i> do <i>Space Shooter</i> (Avaliação, em português).	23
5.5	Resumo do código.	25
7.1	<i>Fitness</i> do primeiro teste do <i>TD</i> (v1).	45
7.2	Taxa de mutação do <i>TD</i> para o primeiro teste (v1).	45
7.3	<i>Fitness</i> do <i>TD</i> (v2) (Avaliação, em português).	46
7.4	Taxa de mutação do <i>TD</i> para cada bateria de testes (2º par dos gráficos).	46
7.5	<i>Fitness</i> do <i>TD</i> (v3) (Avaliação, em português).	46
7.6	Taxa de mutação v3 do <i>TD</i> para cada bateria de testes (3º par dos gráficos).	47
7.7	<i>Fitness</i> de todos os testes do <i>SS</i> (v1).	47

7.8	Trecho do código para cálculo da regressão polinomial.	48
-----	--	----

Sumário

1	Introdução	1
1.1	Motivação	1
1.2	Objetivos	2
2	Gêneros de Jogos e Decisões de Projeto	3
2.1	Jogos com Ondas de Inimigos	3
2.2	Tower Defense	4
2.3	Shooter	5
2.4	Decisões de Projeto	6
3	Plataforma de Desenvolvimento	7
3.1	Nós	8
3.2	Sinais	9
3.3	Cena	9
3.4	Instanciação	10
3.5	Singletons	10
3.6	GDscript	10
3.7	Desenvolvimento dos jogos	11
4	Design dos jogos	13
4.1	Tower Defense	13
4.1.1	O jogo	13
4.1.2	Mecânica de Resistência	16
4.2	Space Shooter	17
5	Algoritmo Genético	21
5.1	O Algoritmo	21
5.1.1	Inicialização	21
5.1.2	Avaliação	23

5.1.3	Seleção	23
5.1.4	Cruzamento	24
5.1.5	Mutação	24
5.1.6	Atualização	25
6	Método de Avaliação do algoritmo	27
6.1	Metodologia	27
6.1.1	Quadrado Latino	27
6.2	Experimentos com o Algoritmo	28
6.2.1	Fitness	28
6.2.2	Taxa de mutação	28
6.3	Desenvolvimento	28
6.3.1	Tower Defense	31
6.3.2	Space Shooter	35
7	Apresentação dos Resultados	37
7.1	Organização dos dados	37
7.2	Dados das Ondas	37
7.2.1	Tower Defense - Ondas Repetidas	38
7.2.2	Tower Defense - Ondas Aleatórias	40
7.2.3	Space Shooter - Ondas Repetidas	41
7.2.4	Space Shooter - Ondas Aleatórias	43
7.3	Fitness dos Jogos	45
7.3.1	Fitness e Mutação	45
7.3.2	Tower Defense - Fitness e Mutação	45
7.3.3	Space Shooter - Fitness e Mutação	47
7.4	Avaliação dos Testes de Versões de Algoritmo Genético	47
7.4.1	Tower Defense - Resultados	48
7.4.2	Tower Defense - Comparação	55
7.4.3	Space Shooter - Resultados	57
7.4.4	Space Shooter - Comparação	63
8	Considerações Finais	65
8.1	Análise dos Testes	65
8.2	Trabalhos Futuros	68
8.3	Conclusão	68

Apêndices

A	Moda das Ondas no Tower Defense para versão v1	71
A.1	Torres Verdes	73
A.2	Torres Vermelhas	77
A.3	Torres Verde + Vermelha	81
A.4	Torres Vermelha + Verde	85
B	Moda das Ondas no Tower Defense para o v2	89
B.1	Torres Verdes	91
B.2	Torres Vermelhas	95
B.3	Torres Verde + Vermelha	99
B.4	Torres Vermelha + Verde	103
C	Moda das Ondas no Tower Defense para a versão v3	107
C.1	Torres Verdes	109
C.2	Torres Vermelhas	113
C.3	Torres Verde + Vermelha	117
C.4	Torres Vermelha + Verde	121
D	Moda das Ondas no Space Shooter para versão v1	125
D.1	Nave Parada com Disparo Amarelo	126
D.2	Nave Movendo com Disparo Amarelo	129
D.3	Nave Parada com Disparo Vermelho	132
D.4	Nave Movendo com Disparo Vermelho	135
E	Moda das Ondas no Space Shooter para versão v3	139
E.1	Nave Parada com Disparo Amarelo	140
E.2	Nave Movendo com Disparo Amarelo	143
E.3	Nave Parada com Disparo Vermelho	146
E.4	Nave Movendo com Disparo Vermelho	149

Referências	153
--------------------	------------

Capítulo 1

Introdução

Jogos digitais são produtos contemporâneos utilizados para obter experiências lúdicas em tempos de lazer, e podem oferecer finalidades educacionais ou terapêuticas, por oferecerem um ambiente seguro onde se propicia diversão, distração e entretenimento (KENT, 2001).

A forma como jogos digitais disponibilizam tais características não permaneceu estática, com inovações conceituais e tecnológicas que acompanham e estimulam a evolução técnica da computação (REIS e CAVICHIOLLI, 2014). Através do uso de orientação a objetos e inteligência artificial, é possível desenvolver um sistema que demonstra um algoritmo genético em evolução contra um jogador.

1.1 Motivação

O desenvolvimento de jogos digitais utiliza diversas técnicas de múltiplas áreas do conhecimento, como arte e design, além da computação (BALDWIN, 2021), assim como propicia e impulsiona o desenvolvimento das mesmas (TSANG, 2021). Tal confluência de várias áreas de conhecimento, assim como a disponibilização de um laboratório para o avanço de técnicas computacionais - devido a necessidades específicas como qualidade visual, desempenho de sistemas e orientação a objeto - sempre visando o entretenimento do público alvo torna a área interessante para a utilização do conhecimento obtido durante a graduação em ciência da computação. Em jogos conhecidos como *single player*, onde um jogador humano concorre contra inimigos automatizados, existe um gênero onde grupos de oponentes atacam o jogador em turnos, chamados de ondas (conhecido como *waves*). Em geral, tais jogos implementam esse comportamento utilizando fases, onde cada onda se torna mais forte para proporcionar um desafio crescente ao jogador, de forma pré-programada se repetindo sempre que um novo jogo é iniciado.

Gerar de maneira procedural os inimigos pode proporcionar um desafio mais variado e desafiador a quem está jogando (SHAKER *et al.*, 2016), mantendo o *design* das fases dos jogos, pois seria relativamente simples gerar inimigos baseados nas opções que um jogador fez e tornar a onda sempre mais forte. O jogo Spelunky DEREK e ANDY (2008) demonstra como geração procedural de elementos - neste caso suas fases - pode deixar a experiência mais

divertida e interessante, utilizando inteligência artificial ao invés de somente aleatorizar elementos (JENSEN, 2020).

1.2 Objetivos

Considerando os pontos apresentados, buscou-se estudar a viabilidade de algoritmos genéticos serem generalizados para jogos com ondas de inimigos - um [Tower Defense](#) e um [Top-Down Shooter](#) - e analisar a performance obtida. O sistema para a geração de ondas deve ser adaptativo ao estilo do jogador, detectando mudanças de estratégia e respondendo com alterações nas ondas, proporcionando maior desafio e variação nos inimigos, mas sem a necessidade de conhecimento profundo de mecânicas específicas do jogo ou a estratégia do jogador, que seriam mais adequadas a uma implementação pré-determinada na geração de inimigos.

Capítulo 2

Gêneros de Jogos e Decisões de Projeto

Existe uma infinidade de estilos de jogos que necessitam de oponentes gerados automaticamente contra o jogador, contudo, considerando o algoritmo genético que seria desenvolvido e a restrição do escopo de desenvolvimento do mesmo, decidiu-se por utilizar jogos com ondas de inimigos. Desta forma, o algoritmo poderia considerar a onda como uma população e separar candidatos apropriados para gerar a próxima onda.

2.1 Jogos com Ondas de Inimigos

Neste estilo de jogo, basicamente, grupos com quantidades finitas de inimigos atacam o jogador, seguindo uma sequência crescente de dificuldade. Em geral as ondas são previamente programadas de acordo com a escala de dificuldade e história que o desenvolvedor deseja demonstrar, mas que podem tornar as partidas repetitivas e facilmente contra-atacadas por jogadores que conseguem abstrair o raciocínio de progresso que foi programado. Ao utilizar um algoritmo genético para produzir as ondas de inimigos, seria possível surpreender o jogador e se adaptar a ele, aumentando a diversidade estratégica e diminuindo a monotonia do jogo.

2.2 Tower Defense

Tower defense é um gênero de jogo onde o jogador deve focar na defesa de território, posicionando defesas limitadas por um recurso - geralmente com a compra de torres que atacam o inimigo - em um caminho pelo qual ondas de oponentes passam, buscando destruir a base do jogador. Pode se citar como exemplo de jogo desse gênero o *game Kingdom-rush* (STUDIO, 2011) que encontra se na Figura 2.1. Para impedir que o inimigo percorra o mapa completamente, as torres devem ser posicionadas para maximizar o tempo de ataque e minimizar o custo, levando em consideração que torres e os inimigos podem apresentar habilidades diferentes como maior resistência ou tiros com maior ou menor dano (ou velocidade), cabendo ao jogador determinar como utilizar os recursos disponíveis.

Existem jogos que apresentam a possibilidade de melhoria das defesas utilizando recursos acumulados ao eliminar oponentes, assim o jogador pode conseguir alterar e aprimorar a estratégia durante as fases da partida. A seleção e escolha do posicionamento das torres é a estratégia essencial do jogo.



Figura 2.1: Kingdom Rush. Obtida de <https://www.ironhidegames.com/Games/kingdom-rush>

2.3 Shooter

Um *shooter game* é um subgênero de jogos de ação, inspirado pelos jogos de fliperama, por exemplo o jogo *Spacewar* (STEVE, 1962), e se estabeleceu com o jogo *Space Invaders* (NISHIKADO, 1978), presente na Figura 2.2. Ambos os jogos funcionam de maneira semelhante, onde uma onda de inimigos deve ser impedida de eliminar o jogador, este podendo se esquivar e atirar neles, sendo que os oponentes também podem dispor de tiros e movimento para causar dano.

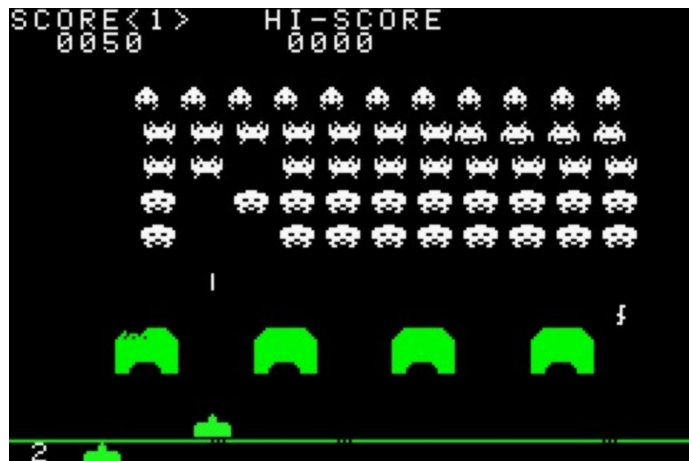


Figura 2.2: *Space Invaders*. Obtida de <https://www.gamekult.com/jeux/space-invaders-17318/test.html>

Dentre os elementos comuns ao gênero, existem variações sobre a capacidade do jogador e dos inimigos se moverem no espaço 2D, onde no *Space Invaders* (NISHIKADO, 1978), o jogador possui somente movimento horizontal, enquanto no *Asteroids* (LYLE e ED, 1979), presente na Figura 2.3, o usuário possui movimentação livre em toda área disponível, mas os oponentes não disparam.

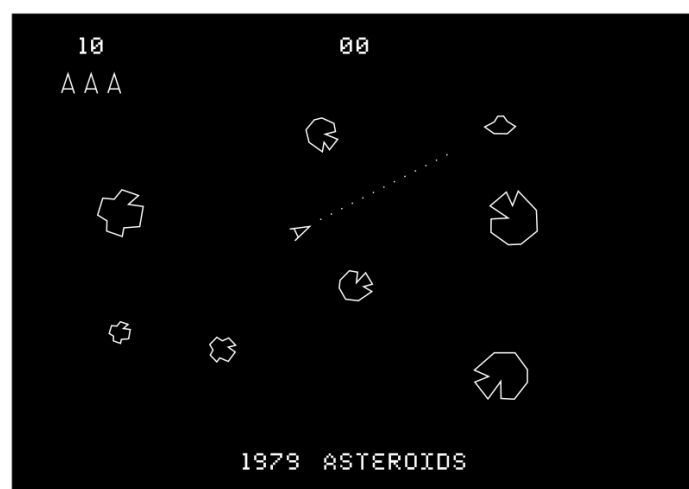


Figura 2.3: *Asteroids*. Obtida de <https://openclipart.org/detail/281726/asteroids-video-games-1979>

2.4 Decisões de Projeto

Considerando que são jogos populares e com jogabilidade simples, decidiu-se desenvolver os dois estilos, considerando a simplicidade de seus objetivos em comum, centrados na sobrevivência por meio da habilidade do jogador de se proteger dos inimigos que atacam em turnos, e a possibilidade dos mesmos serem eliminados à distância.

Ao mesmo tempo, enquanto o *Tower Defense* se apresenta de maneira estática - as torres não se movem, e o jogador não pode alterá-las durante a onda - o *Space Shooter* permite movimentação da nave, logo a habilidade motora e reflexos de quem está jogando são fatores na sobrevivência, e seria interessante verificar como essa característica afeta a evolução do algoritmo genético.

Para desenvolver o projeto decidiu-se utilizar ferramentas *Open Source* sempre que possível, a *Godot Game Engine*, versão 3.3.3; a plataforma de versionamento *GitHub*; a comunicação do grupo se concentrou no serviço *Discord*; a monografia foi desenvolvida de maneira colaborativa na plataforma *OverLeaf*; o *Jupyter Notebook* foi utilizado para análise de dados; e o editor de imagens *GIMP* para edição de imagens.

Capítulo 3

Plataforma de Desenvolvimento

Para auxiliar na produção de jogos existem diversos motores de jogo ou *Game Engines*; ferramentas que dispõem de recursos e bibliotecas para auxiliar no desenvolvimento. Essas ferramentas oferecem estruturas e métodos para simulação de interações físicas, lidar com a entrada dada pelo usuário, dentre outras funcionalidades. Assim são de grande ajuda para o desenvolvimento do jogo como um todo. A *Godot Engine* é uma *game engine* de código aberto, usada no desenvolvimento de jogos 2D e 3D, disponibilizada gratuitamente sob *MIT License*. Outros exemplos de *engine* incluem *Unity*¹, *Unreal Engine*² e *RPG Maker*³.



Figura 3.1: *Godot Game Engine*.⁴

Nesta plataforma, um jogo é um conjunto de cenas composto por nós agrupados em uma estrutura de árvore, estrutura de dado formada por um grafo conexo acíclico. Um nó é a unidade mais básica para construção de elementos, com diferentes propósitos e propriedades, por exemplo, sons, imagens (*sprites*) e câmera. Mesmo com diferentes características e finalidades, todos os nós possuem um nome, variáveis, possibilidade de executarem código (*script*) contendo métodos, herança de funcionalidades ou *script* de outros nós e emissão de sinais de controle para elementos da cena.

¹ <https://unity.com/pt>

² <https://www.unrealengine.com/en-US/>

³ <https://www.rpgmakerweb.com/>

⁴ https://godotengine.org/themes/godotengine/assets/press/logo_large_color_light.png

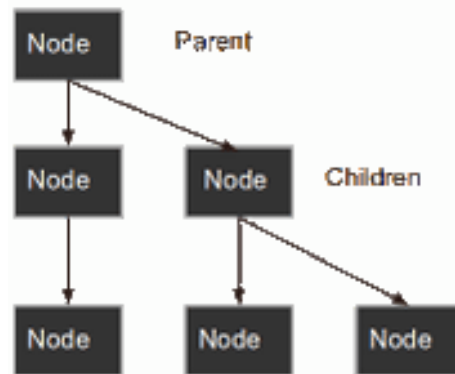


Figura 3.2: Árvore no Godot. ⁵

3.1 Nós

O *Godot* oferece alguns nós específicos para o uso de simulações físicas, fornecendo detecção de colisões entre objetos e diferentes reações caso ocorra o contato. Existem 4 nós específicos para tal função, dependendo do comportamento desejado determinado nó pode ser mais apropriado: *StaticBody2D*, *RigidBody2D*, *KinematicBody2D* e *Area2D*.

Todo nó de colisão/física necessita de um filho do tipo *Shape2D* para detectar colisões, pois este fornece a geometria da região que detecta as colisões. Estas podem ocorrer com outros nós ou com as fronteiras da área de jogo. O nó *Area2D* detecta e emite sinais com a presença de outros nós ao seu redor permitindo a programação de uma reação, por exemplo uma esquiva ou animação.

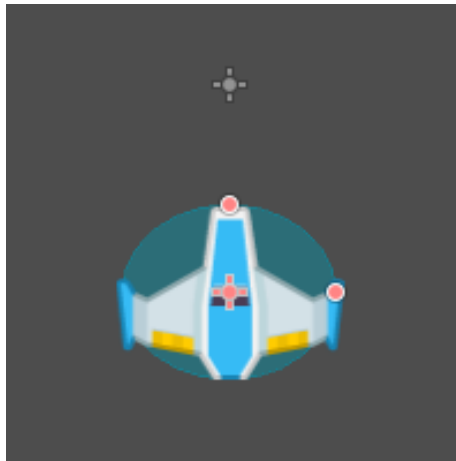


Figura 3.3: Utilizando nós *CollisionShape2D* e *Area2D*. Imagem obtida do editor de jogo.

Na Figura 3.3, feito para o jogo *Space Shooter*, temos um uso dos nós citados, onde a região em azul sobrepõe a imagem da nave. Esta é um nó do tipo *CollisionShape2D* responsável por detectar colisões com outros nós, que no jogo são os adversários. A nave por sua vez é um nó do tipo *Area2D* contendo um *sprite* da nave, e quando os nós

⁵ https://docs.godotengine.org/en/stable/getting_started/step_by_step/scenes_and_nodes.html

colidirem, ocorre a emissão de um sinal que terá um comportamento programado, no caso a perda de "vida" do jogador. O conjunto de nós presentes formam uma cena, nesse caso o *Player*.

Nós podem ser agrupados, e o grupo formado atua em conjunto no jogo, conforme a Figura 3.2. O *Tower Defense* e o *SpaceShooter* se aproveitaram de tal característica para a geração dos inimigos e gerenciamento das ondas.

3.2 Sinais

O *Godot* fornece um sistema de sinais, emitidos e detectados por nós, pré-definidos pelo *engine* ou customizados pelo desenvolvedor, que podem servir de gatilhos ou conexões para outras funções em outros nós dentro da hierarquia fornecida pela árvore da cena. As características dos sinais permitem sua utilização em botões na interface, ações em colisões entre nós, entre outras possibilidades.

3.3 Cena

A cena no *Godot* é composta de nós organizados segundo uma hierarquia de árvore, onde um nó é a raiz, e seus filhos, que podem ser pais de outros nós, sempre controlados pelo nó raiz, conforme a Figura 3.4, cena feita para o jogo *Space Shooter*. Cenas são objetos independentes, podendo representar somente um elemento do jogo - por exemplo os inimigos, o jogador - ou partes completas dele - como um menu, uma fase.

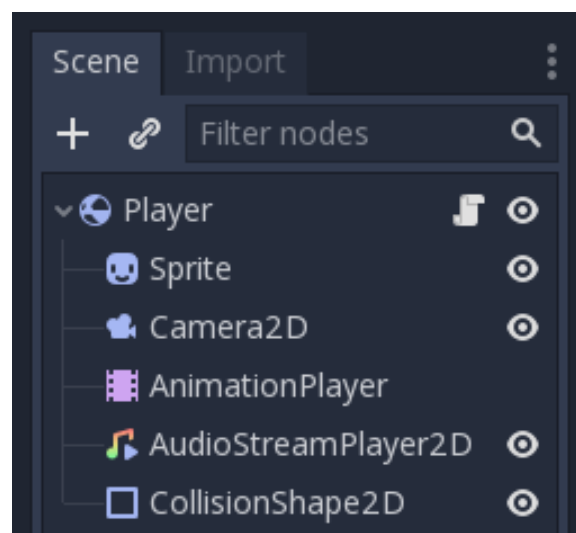


Figura 3.4: Árvore formada por nós no Godot. Em *Player* um ícone indica a presença de um script vinculado ao nó. Imagem obtida da documentação em https://docs.godotengine.org/en/stable/getting_started/step_by_step/scenes_and_nodes.html

3.4 Instanciação

As cenas de um jogo podem ser inicializadas como objetos filhos de uma outra cena, formando uma instância. Este recurso é usado frequentemente nos jogos desenvolvidos, no *Tower Defense* a fase gera instâncias das cenas das torres e dos inimigos, no *Space Shooter* são instanciados o jogador e os asteróides. As instanciações são feitas de maneira dinâmica durante o jogo, sendo carregadas pelos *scripts* e esperando o *input* do jogador, para a instanciação de um novo objeto que é inserido na árvore de execução. É possível observar o uso da instanciação na Figura 3.5, onde a partir do *input* do jogador, projeteis são postos na cena em execução. Cada projétil é uma instanciação, ou objeto, de uma cena feita anteriormente para ser os tiros vistos no jogo.



Figura 3.5: Instanciação de tiros no *Space Shooter*. Imagem retirada do próprio jogo.

3.5 Singletons

Singletons fornecem a possibilidade de acesso global a métodos e variáveis de outras classes no *Godot*. Semelhante aos sinais, os *singletons* oferecem comunicação entre cenas distintas, relevantes no projeto pois permitiu a implementação do algoritmo genético, que necessita de acesso a diversas variáveis dos jogos, como os inimigos, dano causado, entre outros.

3.6 GDscript

Os *scripts* citados são escritos em *GDscript*, linguagem de programação dinâmica de alto nível desenvolvida especificamente para a *Godot Engine*, similar a linguagem de programação *Python* e *Lua*, porém com otimizações e tipos de dados específicos, altamente integrado à própria *engine* (LINIETSKY e MANZUR, 2018a). O editor fornecido integra diversas funcionalidades de um ambiente de desenvolvimento avançado em sua interface, como editor visual 2D e 3D, editor de texto, conexões de sinais, depurador, visualização da

árvore de execução durante o desenvolvimento e em tempo real durante a execução, que facilitam o início da produção de jogos.

3.7 Desenvolvimento dos jogos

Para facilitar a familiarização com o ambiente de desenvolvimento e acelerar o foco no algoritmo genético, foram utilizados diversos recursos disponíveis para a produção dos jogos. Desde repositórios de *sprites* como [KENNEY \(2021\)](#) a tutoriais de *Godot*, sendo o *Space Shooter* baseado no tutorial de [LINIETSKY e MANZUR \(2018b\)](#), disponível na documentação da *engine*, enquanto o *Tower Defense* foi baseado no tutorial disponibilizado pelo [CENTER \(2021\)](#). Tanto [KENNEY \(2021\)](#) como [LINIETSKY e MANZUR \(2018b\)](#) possuem licenças que permitem o uso público ou equivalente, além de serem gratuitas.

Capítulo 4

Design dos jogos

O desenvolvimento dos jogos foi focado em características comuns entre si, para possibilitar o uso do mesmo algoritmo genético, minimizando a necessidade de ajustes no código. A implementação inicial foi feita no *Tower Defense*, e em seguida no *Space Shooter*.

4.1 Tower Defense

4.1.1 O jogo

O jogo *Tower Defense* ¹ consiste em mapa com dois caminhos a serem percorridos por tanques inimigos e oferece ao jogador duas torres para impedir que os tanques completem seu trajeto. O início do percurso é na esquerda da tela, terminando na direita, conforme visto na Figura 4.1.

Antes de apertar o botão *Play*, o jogador pode colocar quantas e quaisquer torres desejar em áreas livres do mapa. Iniciando a partida, uma onda de 12 tanques começa a percorrer o caminho, inicialmente 6 para cada percurso, conforme a Figura 4.4. Posteriormente, o algoritmo genético gerencia a rota. Ao lado do botão *Play* estão botões que aceleram a execução do jogo em duas vezes e oito vezes, para facilitar a coleta de dados para os testes.

As duas torres disponíveis oferecem diferentes áreas de alcance para detecção de tanques, apresentadas nas Figuras 4.2 e 4.3 e quantidade de dano que podem causar, como esta descrito na Tabela 4.1. Cada inimigo que atinge o objetivo produz dano ao jogador, que possui 100 pontos de vida, mostrado pela barra *HP* na tela. A região também mostra os recursos monetários disponíveis ao jogador - que não está implementado - e a onda atual; o jogo somente termina quando acaba o *HP* do jogador.

¹ Repositório contendo o jogo: <https://github.com/raktanaka/tccTD> - 21/12/2021

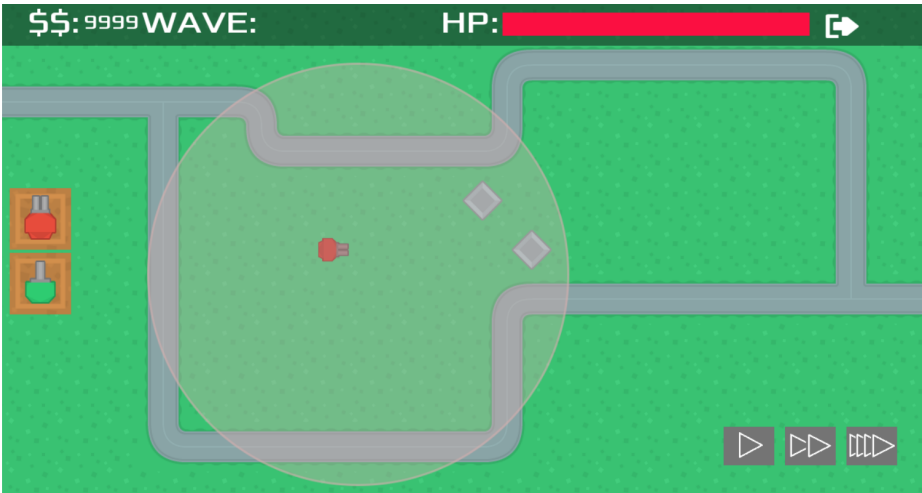


Figura 4.3: Área de detecção torre vermelha. Imagem retirada do próprio jogo.

	velocidade de tiro	dano	alcance
Torre Verde	55	25	350
Torre Vermelha	70	15	550

Tabela 4.1: Dano das torres no Tower Defense.

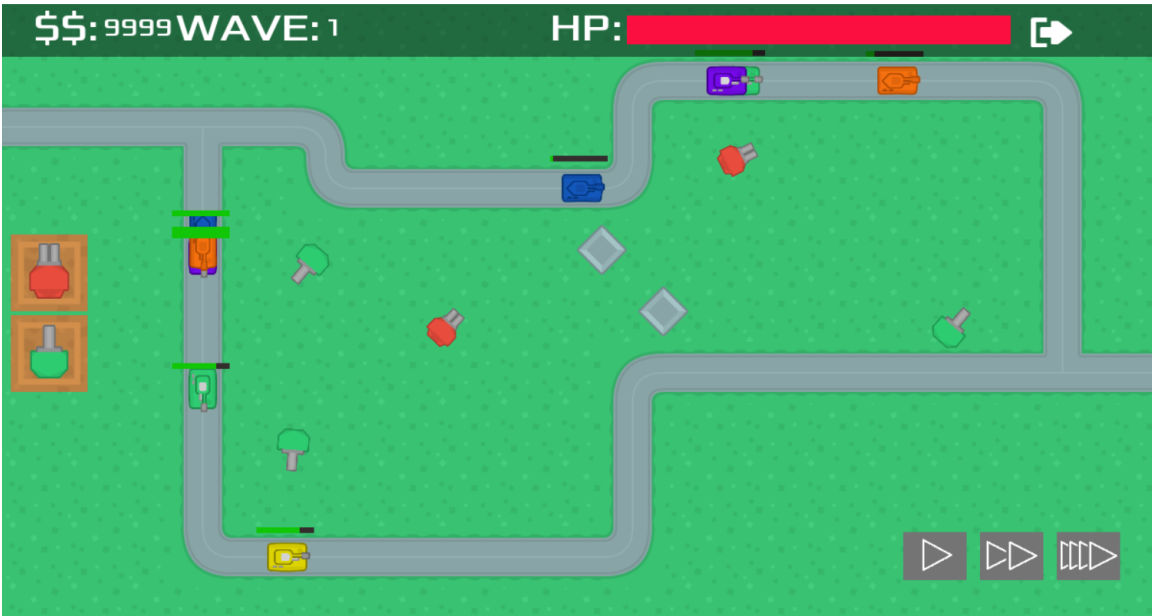


Figura 4.4: Exemplo de uma execução do jogo. Imagem retirada do próprio jogo.

Existem 6 tipos de tanques diferentes, descritos na Figura 4.5. Cada tipo de tanque possui 2 características, sendo elas velocidade e dano causado ao jogador, cujo os valores variam conforme o tipo, visto a Tabela 4.2.

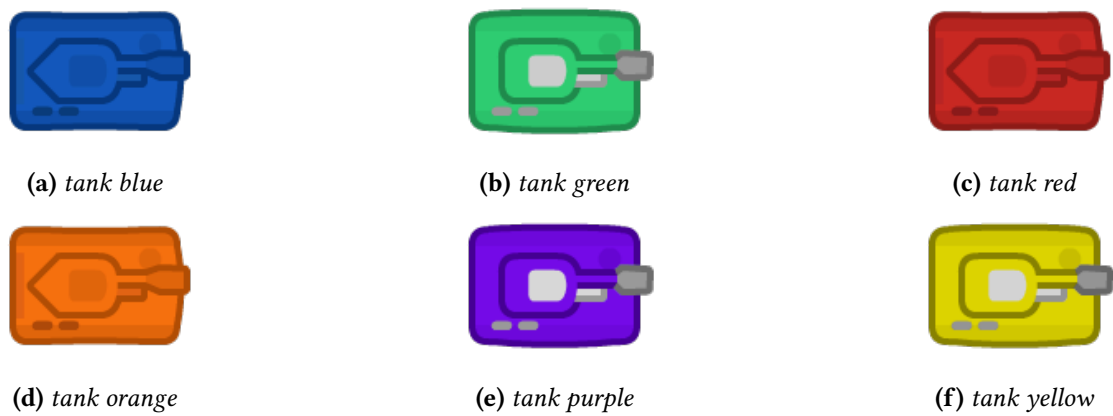


Figura 4.5: Tipos de tanques

	velocidade	dano
tank blue	55	55
tank green	70	45
tank red	80	15
tank orange	120	5
tank purple	90	15
tank yellow	150	5

Tabela 4.2: Dano de cada tanque no Tower Defense

Ficam definidas como Norte e Sul as rotas disponíveis, na Figura 4.6.

4.1.2 Mecânica de Resistência

No jogo de *Tower Defense* também foi adicionado um sistema de resistência nos tanques, cada Tanque possui resistência contra Torres de **mesma cor** e recebem dano reduzido pela metade dos mesmos. Segue um pseudocódigo da implementação do tanque ao receber dano de uma torre.

Programa 4.1 Sistema de Resistência TD.

```
1  ▷ x = tanque que recebe o dano
2  ▷ damage = quantidade total de dano recebido
3  ▷ color_tower = cor da torre que disparou no tanque.
4  func on_hit(x, damage, color_tower):
5  ▷ color(x) = cor do tanque x
6  ▷ hp(x) = hp atual de x.
7    se color(x) = color_tower:
8      hp(x) -= damage / 2
9
10   senao:
11     hp(x) -= damage
12 fim
```

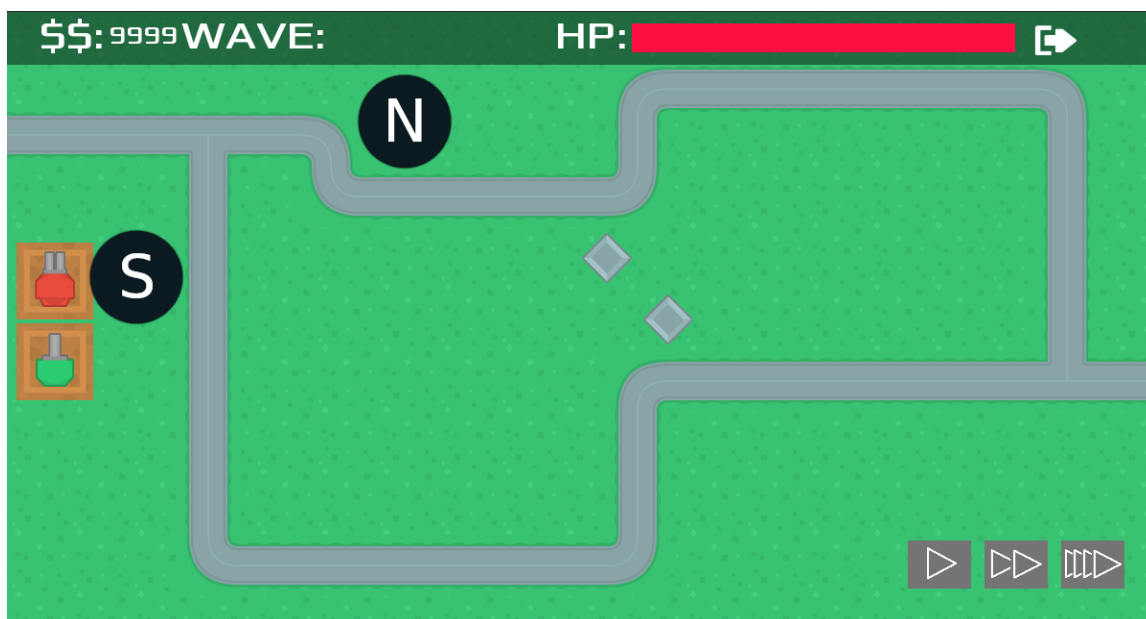



Figura 4.6: Definição das rotas Norte e Sul. Imagem retirada do jogo e editada.

4.2 Space Shooter

O jogo *Space Shooter*² consiste em uma nave que o jogador pode mover livremente em um espaço 2D, onde este deve sobreviver a ondas de asteroides que partem de diferentes posições da tela em sua direção. O jogador além de esquivar, pode eliminar os asteroides por meio de disparos com o *mouse*.

Existem dois tipos de disparos que o jogador pode usar para sua defesa, cada um com características diferentes. No começo do jogo é requisitado que o jogador escolha qual tipo de disparo irá utilizar, conforme as Figuras 4.7 e 4.8.

	Velocidade de Tiro	Dano
Disparo	1200	15
Disparo 1	600	30

Tabela 4.3: Dano dos tiros no *Space Shooter*

Existem dois tipos de *player* automáticos disponíveis para escolha, caso o jogador marque a opção 'yes' na barra inferior, próximo ao texto 'AI', como pode se observar na Figura 4.9. Tais *players* foram desenvolvidos com o intuito de auxiliar no projeto, e na realização dos testes e coleta de dados. Uma delas faz com que a nave transite da direita para esquerda em um intervalo de cinco minutos, a outra opção faz com que a nave fique parada no meio da tela, em ambos atirando no inimigo mais próximo que detectar, de forma a destruir o asteroide que vem em sua direção ou ser atingida.

Existem 6 tipos de asteroides diferentes, presentes na Figura 4.10. Cada tipo de asteroide possui 3 características, sendo elas velocidade, vida ou resistência e dano causado ao jogador,

² Repositório contendo o jogo <https://github.com/RGPRafael/godot> - 21/12/2021

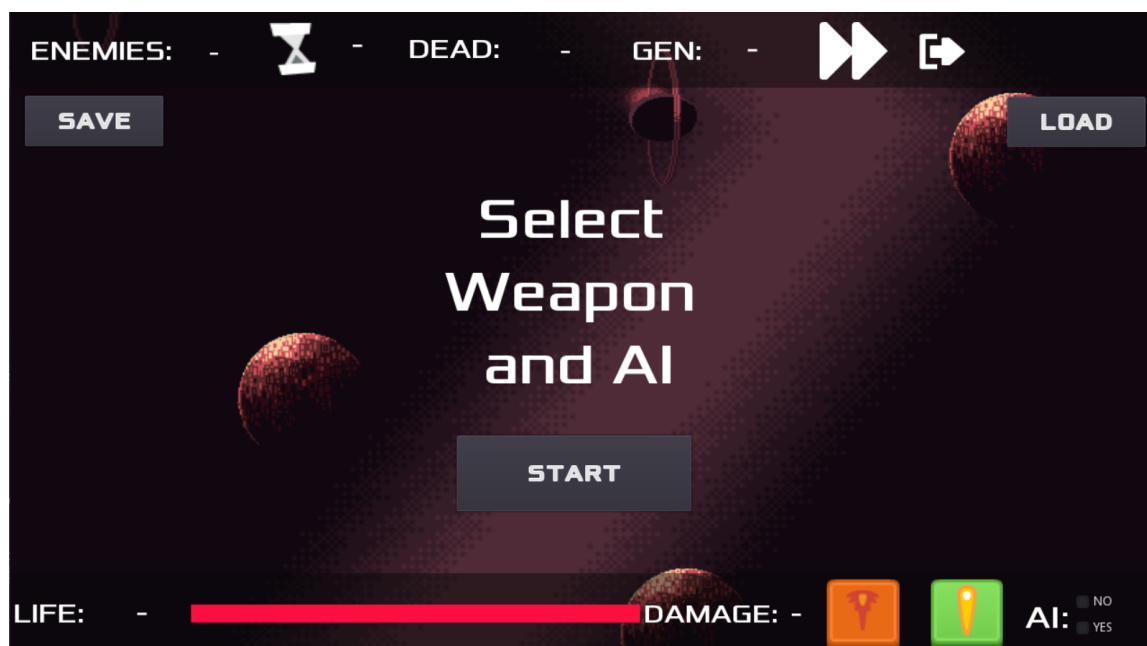


Figura 4.7: Tela inicial. Imagem retirada do próprio jogo.



(a) Disparo

(b) Disparo1

Figura 4.8: Disparos

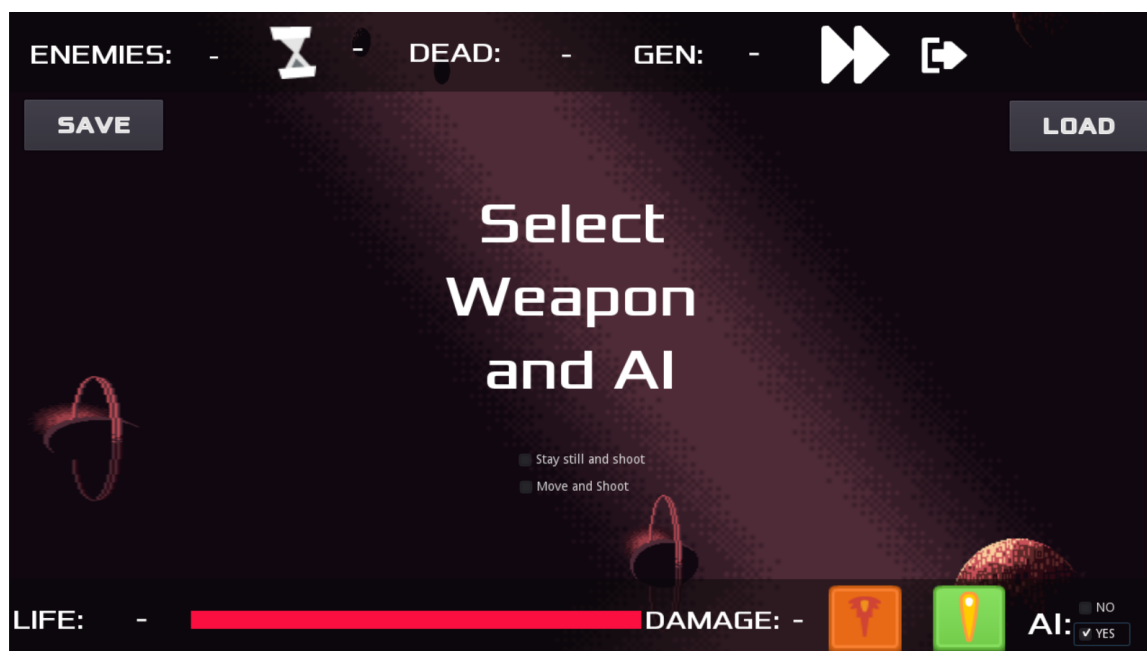


Figura 4.9: Opções de Player Automático. Imagem retirada do próprio jogo.

cujo os valores variam conforme o tipo, presentes na Tabela 4.4.

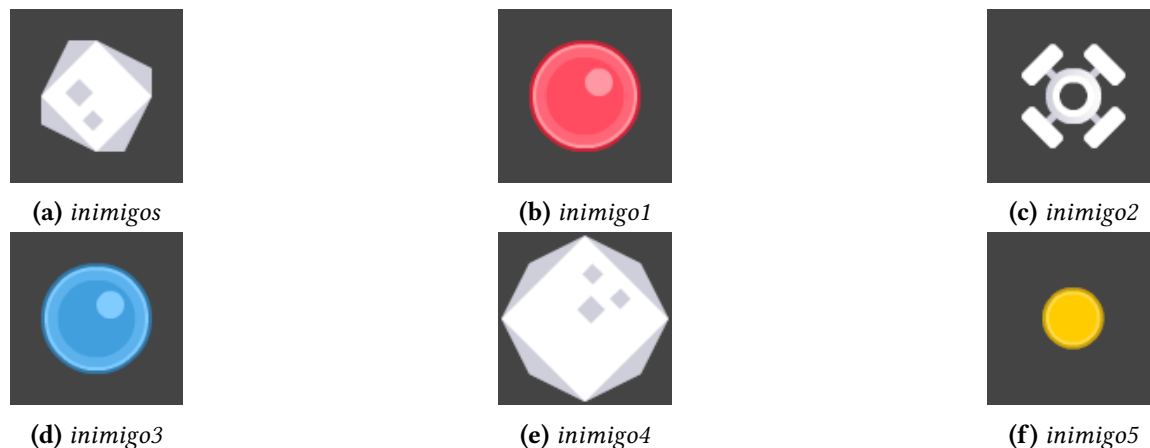


Figura 4.10: Tipos de asteroides

	Velocidade	Dano	Vida
inimigos	500	20	50
inimigo1	350	45	30
inimigo2	470	50	40
inimigo3	320	60	40
inimigo4	550	20	60
inimigo5	700	10	120

Tabela 4.4: Velocidade, dano e vida de cada asteroide no Space Shooter

Após feitas as escolhas sobre o tipo de disparo que será utilizado e se o jogador vai mover a nave ou não, basta apertar o botão 'Start' para início do jogo. O jogador assim deve sobreviver às ondas de inimigos que aparecem de diferentes posições da tela.

Caso seja atingido, a quantidade de dano que sofreu é exibida no campo 'Damage'. Por sua vez, quantidade de inimigos que apareceram enquanto o jogo ainda se desenrola é mostrada na barra superior na extrema esquerda, assim como outras informações como numero de asteroides que foram mortos e o números de gerações que foram enfrentadas. Há um botão para acelerar a velocidade do jogo, e outro que volta ao começo do jogo, caso o jogador não queria continuar ou tenha se arrependido das configurações iniciais que fez anteriormente, um exemplo de uma rodada do jogo encontra-se na Figura 4.11.

O jogador começa com a sua barra de vida no valor total igual a 100. Caso chegue a zero 3 vezes o jogador morre e o jogo volta à tela inicial onde as configurações sobre a escolha de arma, por exemplo, devem ser feitas de novo. Ficam definidos os locais de spawn de 1 até 6 na Figura 4.12

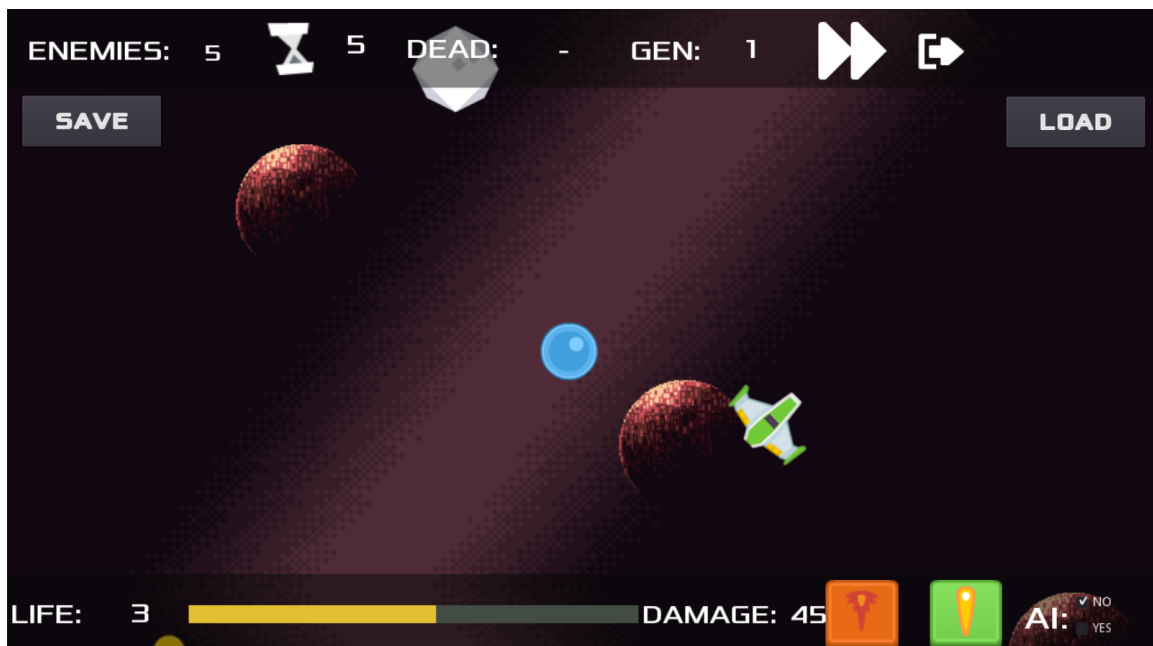


Figura 4.11: Exemplo de uma rodada do jogo. Imagem retirada do próprio jogo.

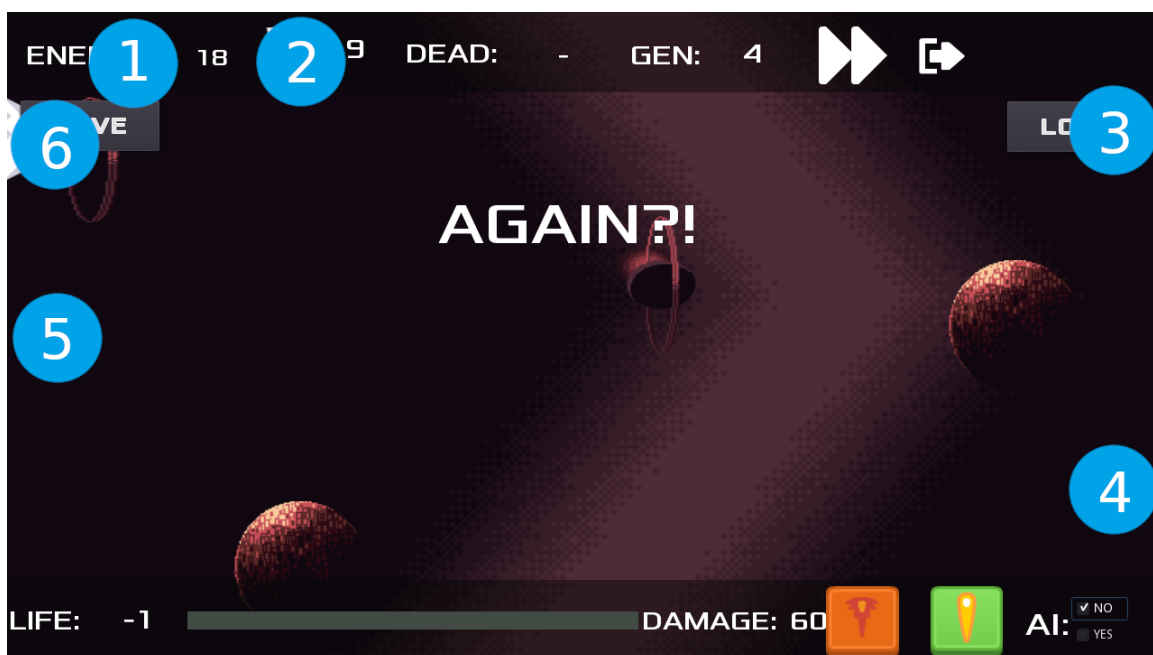


Figura 4.12: Posições possíveis onde um asteroide pode surgir. Imagem retirada do jogo e editada.

Capítulo 5

Algoritmo Genético

Algoritmo genético é uma técnica de evolução computacional desenvolvida para o estudo de comportamento adaptativo (A.E. EIBEN e SMITH, 2015). De maneira simplificada, o algoritmo cria um conjunto de diferentes resoluções para um dado problema e, a partir da avaliação desse conjunto, seleciona as melhores soluções.

Esse processo imita a ideia de seleção natural proposta por Darwin (A.E. EIBEN e SMITH, 2015), onde os indivíduos de uma população disputam por recursos para sobreviverem. Aqueles que sobrevivem conseguem passar seus genes para a prole e a população seguinte será mais adaptada ao ambiente e com mais chances de reprodução, passando as características que permitiram sua sobrevivência para as gerações seguintes. Portanto, um algoritmo genético é composto das seguintes etapas: Inicialização, Avaliação, Seleção, Cruzamento (*Cross over*), Mutação e Atualização (Figura 5.1).

5.1 O Algoritmo

No projeto, cada indivíduo é um inimigo criado pelo jogo e a população se resume a uma "wave" (onda de inimigos). Os genes representam características importantes dos inimigos, como: Tipo (TD e SS), Rota (TD), Local de *Spawn* (SS), conforme esboçado por MALLAWAARACHCHI (2017) e GAD (2018).

5.1.1 Inicialização

Na primeira etapa, inicialização, são codificados os genes para que sejam submetidos ao processo de avaliação.



Figura 5.1: Estrutura de um algoritmo genético.

Programa 5.1 Inicialização do *Space Shooter* com tipo de inimigo e local de surgimento.

```

1  var population = [[inimigos, Vector2(90,-50) ],
2                      [inimigo1, Vector2(90,-50)],
3                      [inimigo2, Vector2(90,-50)],
4                      [inimigo3, Vector2(90,-50)],
5                      [inimigo4, Vector2(90,-50) ],
6                      [inimigo5, Vector2(90,-50)]]
  
```

Programa 5.2 Inicialização do *Tower Defense* com tipo de inimigo e rota.

```

1  var population = [[EnemyRed, 0], [EnemyGreen, 0], [EnemyBlue, 0],
2                      [EnemyYellow, 0], [EnemyPurple, 0], [EnemyOrange, 0],
3                      [EnemyRed, 1], [EnemyGreen, 1], [EnemyBlue, 1],
4                      [EnemyYellow, 1], [EnemyPurple, 1], [EnemyOrange, 1]]
  
```

No *Tower Defense* o algoritmo inicia com uma população de 12 inimigos, 1 de cada

tipo para cada rota; no *Space Shootes* são 1 de cada tipo em cada posição de *spawn*. Uma amostra que pareceu adequada para o propósito do experimento, uma vez que populações pequenas convergem mais rapidamente, economizando tempo para múltiplas execuções, e não tem indícios de serem piores do que largas populações. [HAUPT \(2000\)](#)

5.1.2 Avaliação

A avaliação (ou função *Fitness*) realiza uma análise que estabelece o quão bem os indivíduos da população resolvem o problema proposto. Na natureza, se trata da habilidade de um indivíduo competir contra os demais da população, no algoritmo proposto segue:

Programa 5.3 *Fitness do Tower Defense* (Avaliação, em português).

```

1  FUNCAO fitness_TD (x)  ▷ Dá uma pontuação para indivíduo x
2  ▷ offset (x) = quanto do caminho foi percorrido, float de 0 a 1.
3  ▷ hp (x) = quanto de HP sobrou do inimigo, float de 0 a 1.
4  fit ← (offset (x) + hp (x)) / 2  ▷ média aritmética dos valores de x
5  devolva fit  ▷ Um float com uma pontuação no intervalo [0,1].
6  fim

```

Programa 5.4 *Fitness do Space Shooter* (Avaliação, em português).

```

1  FUNCAO fitness_SS (x)  ▷ Dá uma pontuação para indivíduo x
2  ▷ reached-goal (x) = booleana se acertou o inimigo ou nao
3  se reached-goal
4  fit ← 5
5  senao
6  fit ← 0
7
8  ▷ hp(x) = 5 * quanto de HP sobrou do inimigo (mob), no intervalo [0,1]
9  fit ← (fit + hp(x)) / 10  ▷ média aritmética dos valores de x
10 devolva fit  ▷ Um float com uma pontuação no intervalo [0,1].
11 fim

```

Deve ser ressaltado que o *fitness* é uma característica inerente a cada sistema, com múltiplas possibilidades de avaliação que produzem resultados que podem ser considerados corretos, encontrando máximos locais que satisfazem os problemas propostos.

5.1.3 Seleção

Na etapa de seleção, os indivíduos são selecionados para reprodução com base na sua aptidão obtida anteriormente. O algoritmo seleciona 2/3 dos indivíduos como a melhor abordagem, em ordem decrescente dos valores de *fitness*, com base na natureza (2 pais geram 1 filho). Contudo, os experimentos e consulta a referências, mostraram que a escolha do tamanho da prole não é trivial. ([JANSEN et al., 2005](#))

5.1.4 Cruzamento

No cruzamento, as soluções são recombinadas gerando novos indivíduos, tomando parte dos genes do pai e da mãe (dois indivíduos distintos na população); sendo considerada metade dos genes do pai e da mãe, gerando 1/3 dos indivíduos para a próxima geração. A Figura 5.2 a seguir ilustra o cruzamento, onde *crossover point* representa o ponto em que os genes foram divididos, com os retângulos vermelhos representando características que a IA selecionou de cada um dos pais.

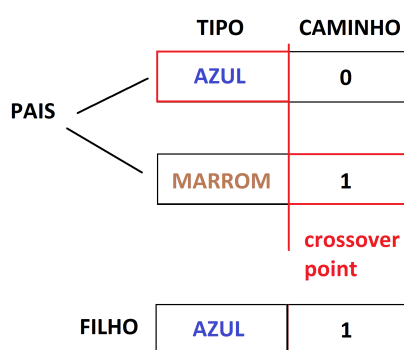


Figura 5.2: Crossover para o caso do TD.

5.1.5 Mutação

A mutação representa a parte do algoritmo que equilibra '*exploitation vs exploration*', ou seja, o quanto o algoritmo deve continuar buscando novas soluções (*exploration*) ou o quanto ele deve continuar tirando vantagem da solução que já encontrou (*exploitation*). No caso do algoritmo genético implementado, a mutação pode ocorrer em um único gene de um ou mais indivíduos da população, adicionando variabilidade ao resultado final. (A. EIBEN e SCHIPPERS, 1998).

No caso do trabalho, como os testes tinham rigor mais determinístico (pela implementação), era esperado que a população final convergisse para algum resultado. Portanto, a taxa de mutação escolhida foi de $\frac{1}{n^{\circ} \text{ de indivíduos na pop}} = \frac{1}{12} \text{ ou } \frac{1}{6}$. Uma taxa que pareceu adequada conforme o artigo de HAUPT (2000).

5.1.6 Atualização

Os indivíduos resultantes serão adicionados na população na etapa de atualização e, caso seja necessário, o processo descrito acontece novamente.

Programa 5.5 Resumo do código.

```

1  FUNCAO start_experiment (pop)  ▷ Cria uma nova população baseada na população
    atual
2  ▷ Quantidade de pais para reprodução
3  num_parents_mating ← population.size() * 2 / 3
4
5  ▷ Função de Avaliação dos pais mais aptos da população
6  fitness ← cal_pop_fitness(population_res)
7
8  ▷ Função de Seleção dos melhores candidatos para a reprodução
9  parents ← select_mating_pool(population, fitness, num_parents_mating)
10
11  ▷ offspring_size [0] = quantidade de filhotes
12  ▷ offspring_size [1] = quantidade de genes
13  ▷ length(pop) = quantidade de individuos em pop
14  ▷ genes(pop) = quantidade de genes de qualquer individuo de pop
15  offspring_size ← [length(pop) - parents.size(), genes(pop)]
16
17  ▷ Função de Cruzamento dos pais para gerar os filhos.
18  offspring_crossover ← crossover (parents, offspring_size)
19
20  ▷ Função de Mutação dos filhos
21  offspring_mutation ← mutation (offspring_crossover)
22
23  ▷ Adiciona os pais que sobreviveram de volta no vetor população
24  para i = 0 até tamanho (parents):
25      new_population.append (parents[i])
26  fim
27
28  ▷ Adiciona a nova prole para a população
29  para i = 0 até tamanho (offspring_mutation):
30      new_population.append (offspring_mutation[i])
31  fim
32
33  devolva new_population
34  fim

```

Capítulo 6

Método de Avaliação do algoritmo

Com o algoritmo funcional, foi necessário buscar maneiras de avaliar seu desempenho para garantir sua eficiência *versus* inimigos gerados aleatoriamente. Foram consideradas as possibilidades de distribuir os jogos e coletar *feedback* de outros jogadores, mas considerando possíveis avaliações subjetivas para os jogos relativamente simples e limitações de tempo decidiu-se por utilizar métodos automatizados para coleta de resultados de dano em sequências de ondas.

6.1 Metodologia

Segundo o Teorema Central do Limite, são necessárias, no mínimo, 30 amostras para que a média das mesmas tenha uma distribuição aproximadamente normal (MAGALHAES e LIMA, 2010). Portanto, foram planejados experimentos nos jogos para geração de 30 amostras de dados em cada modo de jogo, para produção de gráficos com o comportamento das ondas, e desenho em quadrado latino.

6.1.1 Quadrado Latino

A coleta e disposição dos dados teve como objetivo a montagem de um quadrado latino. Um quadrado latino de v linhas por v colunas é um arranjo de v letras latinas na forma de uma tabela, de maneira que cada letra ocorra apenas uma vez em cada linha, e uma vez em uma coluna, (ANGELA e DANIEL, 1999). Por exemplo, a tabela 3 x 3 é um quadrado latino:

A	B	C
B	C	A
C	A	B

Tabela 6.1: Exemplo de quadrado latino

O desenho em quadrado latino é usado em experimentos onde os objetos de estudo

estão submetidos a uma series de tratamentos, onde o tempo decorrido terá influência significativa nas respostas (ANGELA e DANIEL, 1999). A escolha do desenho em quadrado latino também é apropriada dado que os ambientes em que os testes foram feitos não são exatamente iguais, e por existirem dois fatores que podem influenciar nas respostas (CAMPBELL e STANLEY, 1963) - a inteligência artificial e o jogador que está sendo simulado.

6.2 Experimentos com o Algoritmo

Durante o decorrer dos experimentos, notou-se incoerências nos testes realizados pelos códigos produzidos. Em especial, na função *fitness* e na taxa de mutação.

6.2.1 Fitness

A função de avaliação (*fitness*) é passada pelo usuário do código e, portanto, tiveram que ser criadas. No próximo capítulo 7.3, será explicado mais detalhadamente sobre os resultados produzidos com diferentes funções de avaliação (*fitness*) e as tentativas de alcançar outros resultados com base nelas.

6.2.2 Taxa de mutação

A taxa de mutação também pode ser definida pelo usuário. Em particular, para testes pequenos como os feitos neste trabalho, é recomendado um valor entre 5% até 20%. (HAUPT, 2000).

Uma diferente abordagem ainda nesse tópico, é a utilização de taxas de mutação que decrescem conforme o tempo. Possui seus benefícios em testes de natureza mais determinística, já que espera-se que o algoritmo chegue em um ponto de convergência nas populações finais. Este método proporciona maior *exploration* nas gerações iniciais, uma vez que está buscando o melhor resultado sem se prender em pontos de máximo local e maior *exploitation* nas gerações finais, pois estará explorando o melhor resultado que obteve até o momento. (A. EIBEN e SCHIPPERS, 1998)

6.3 Desenvolvimento

Para fazer a coleta dos dados de maneira mais eficiente, os jogos foram adaptados para armazenarem os resultados das ondas. Foram desenvolvidas cenas no *Godot* (menus e fases de teste), como pode se ver nas Figuras 6.1 e 6.2, que executam os testes de maneira determinística e consistente, sem aceitar *input* do jogador. O botão *Test Mode* acessa a interface de testes, enquanto o botão *Play Defaut mode* utiliza o modo de jogo padrão. Todos os resultados são gravados em arquivos ".txt" com nome e formatação pré-definidos para facilitar a análise exploratória dos dados em *Jupyter Notebook* ¹.

¹ <https://github.com/raktanaka/tcc-results> - 21/12/2021

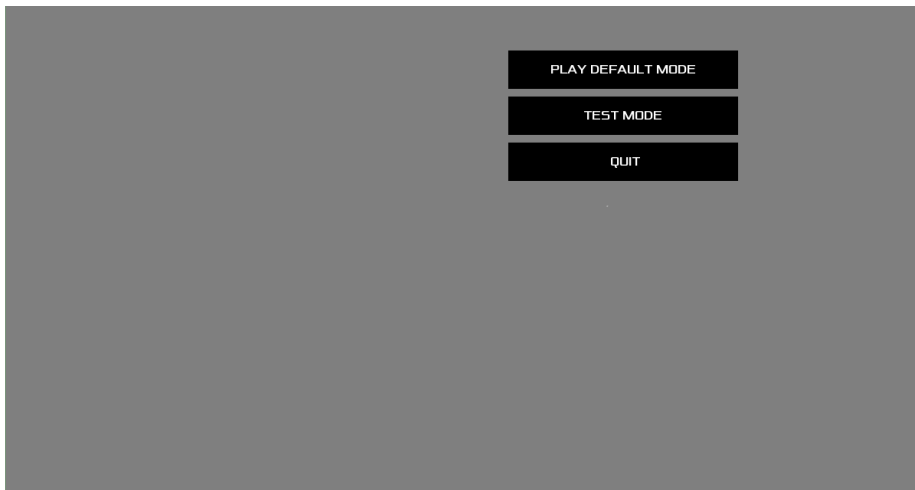


Figura 6.1: Tela de Teste Tower defense. Imagem retirada do próprio jogo.

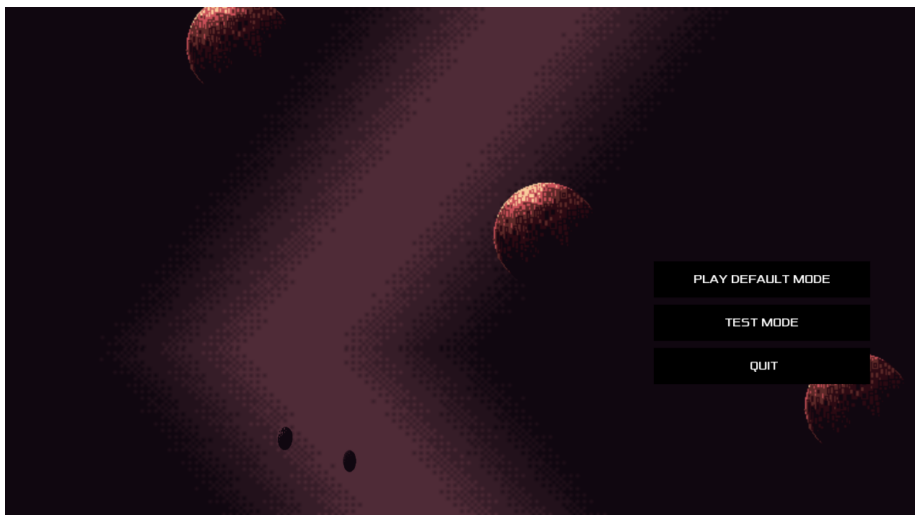


Figura 6.2: Tela de Teste Space Shooter. Imagem retirada do próprio jogo.

O menu *Test Mode* permite escolher as configurações necessárias para a execução de um teste, com condições específicas e pré-definidas para obtenção de dados sobre as ondas geradas e o dano total produzido, o menu *Test Mode* de cada jogo encontram-se nas Figuras 6.3 e 6.4.

O algoritmo genético foi testado (opção *AI* no menu), em conjunto com opções ingênuas nos dois jogos, posto que foram implementadas para ser comparadas com as ondas geradas pelo algoritmo desenvolvido, em vista de observar se seu desempenho. Assim há opções, apresentadas nas Figuras 6.3 e 6.4 onde as ondas podem ser geradas:

- Aleatoriamente - *RANDOM*
- Um de cada oponente - *ONE EACH*
- Modos "repetição", onde todo inimigo é do mesmo tipo



Figura 6.3: Menu de testes Tower Defense. Imagem retirada do próprio jogo.

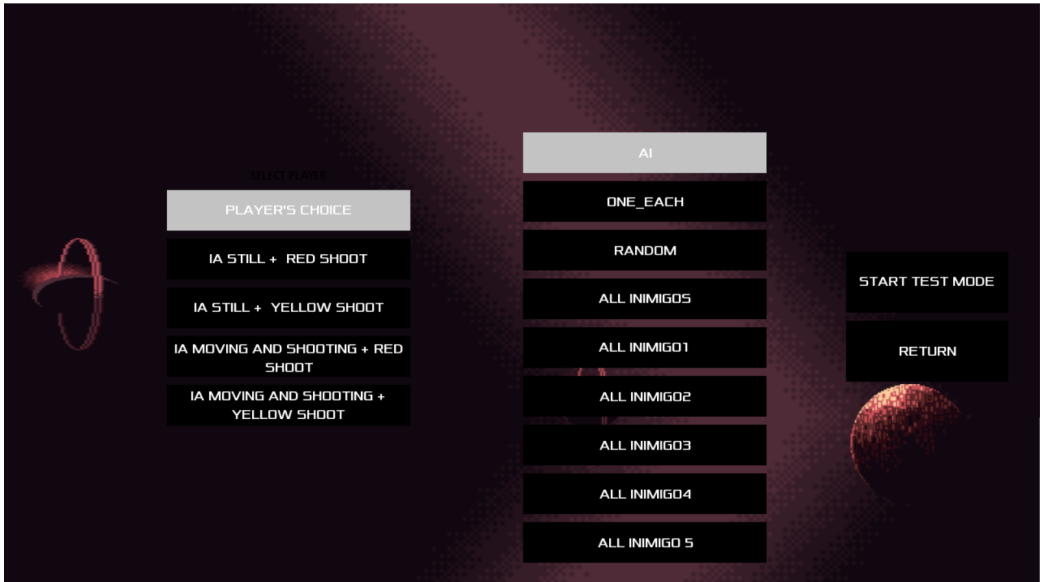


Figura 6.4: Menu de Testes Space Shooter. Imagem retirada do próprio jogo.

6.3.1 Tower Defense

No *Tower Defense*, as ondas repetidas enviam 6 inimigos iguais para cada rota norte e sul, um exemplo de uma onda repetida encontra-se na Figura 6.7 seguindo as rotas que foram já definidas conforme a Figura 4.6. Cada onda repetida envia o mesmo tipo de tanque, cujos atributos estão disponíveis na Tabela 6.3. O modo *One Each* utiliza os 6 tipos em cada rota; e o *Random* aleatoriza completamente a escolha de tipo e rota, utilizando *RandomNumberGenerator* disponibilizada pela *Godot Engine*. Os modos *Random* e *One Each* podem ser vistos nas Figuras 6.5 e 6.6.

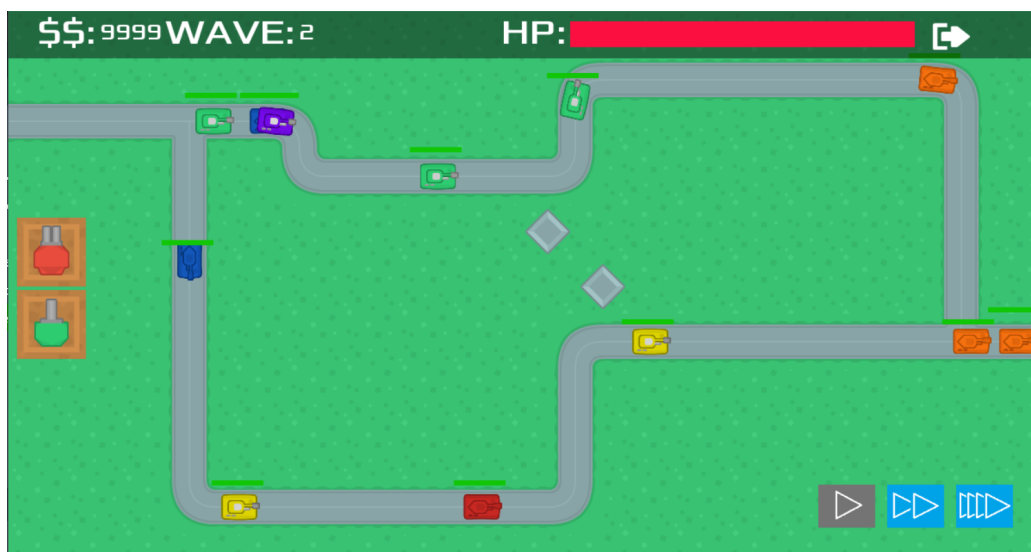


Figura 6.5: Escolha Random em tower defense. Imagem retirada do próprio jogo.

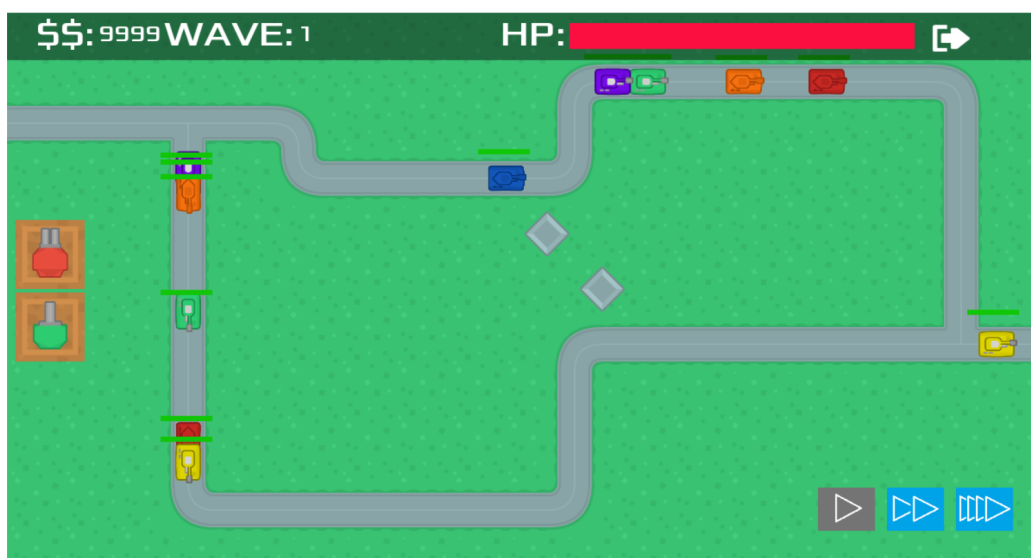


Figura 6.6: Escolha One Each no Tower Defense. Imagem retirada do próprio jogo.

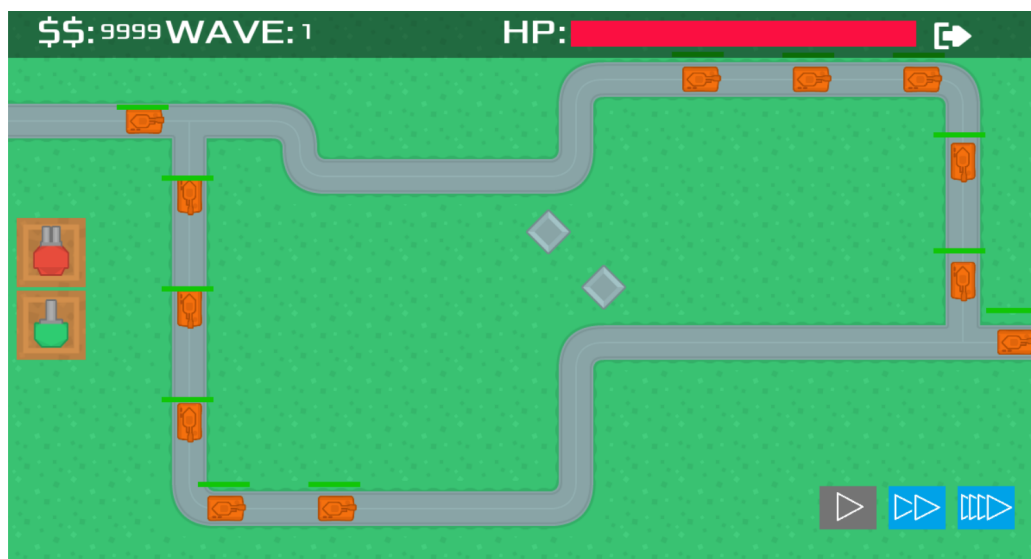


Figura 6.7: Opção All Orange. Imagem retirada do próprio jogo.

Para uniformizar o comportamento do jogador, as torres foram configuradas para melhor controle dos testes, posicionadas duas por rota, onde o alcance das mesmas fica restrito a somente seu caminho, de maneira a alcançar um balanço entre eliminação e sobrevivência de alguns oponentes, visto as torres terem diferentes valores atribuídos com relação a dano e alcance, presentes na Tabela 6.2. Assim são geradas 4 configurações de teste, demonstradas nas Figuras 6.8, 6.9, 6.10, 6.11.

	velocidade de tiro	dano	alcance
Torre Verde	55	25	350
Torre Vermelha	70	15	550

Tabela 6.2: Dano das torres no Tower Defense.

	velocidade	dano
tank blue	55	55
tank green	70	45
tank red	80	15
tank orange	120	5
tank purple	90	15
tank yellow	150	5

Tabela 6.3: Dano de cada tanque no Tower Defense

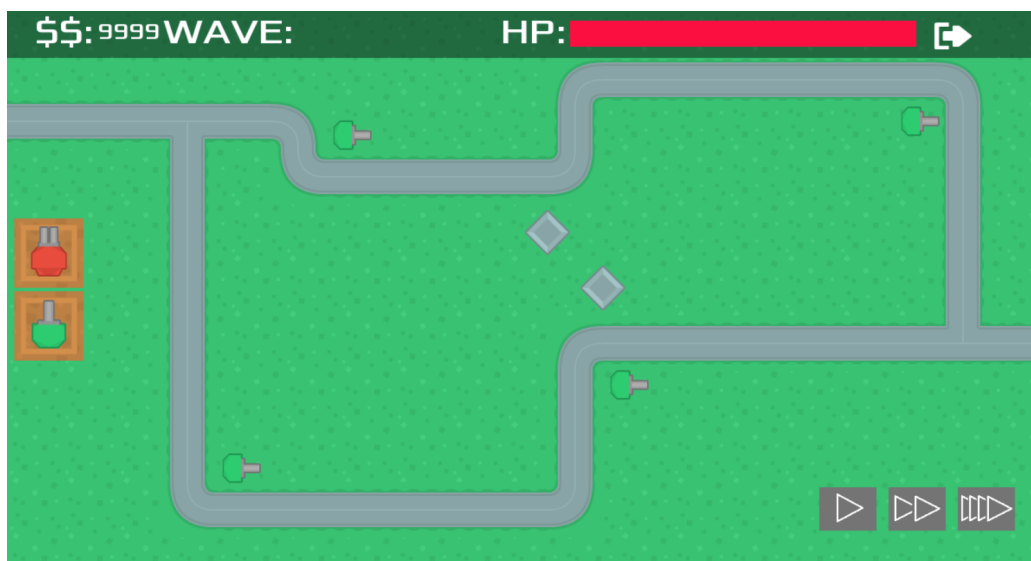


Figura 6.8: *Teste de torres verdes*

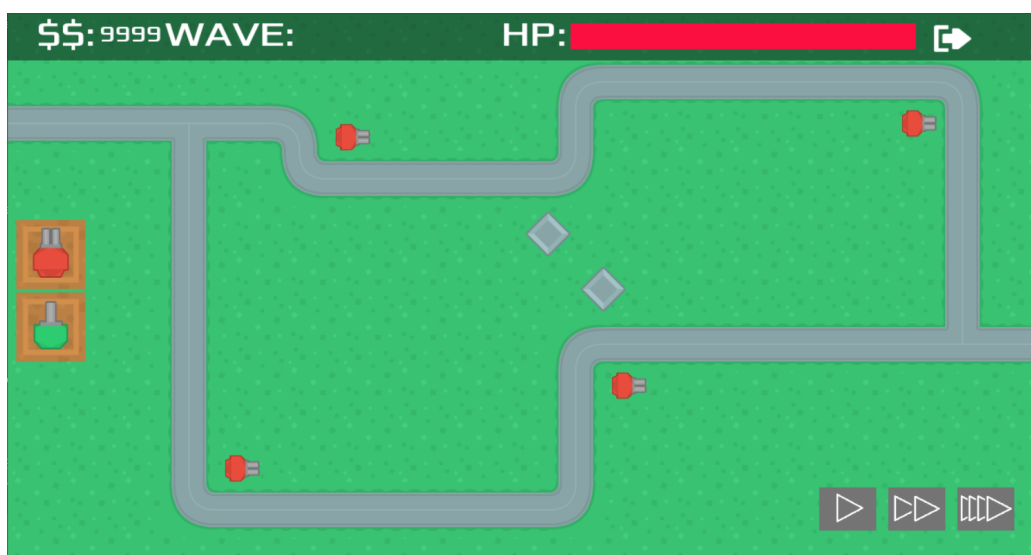


Figura 6.9: *Teste de torres vermelhas*

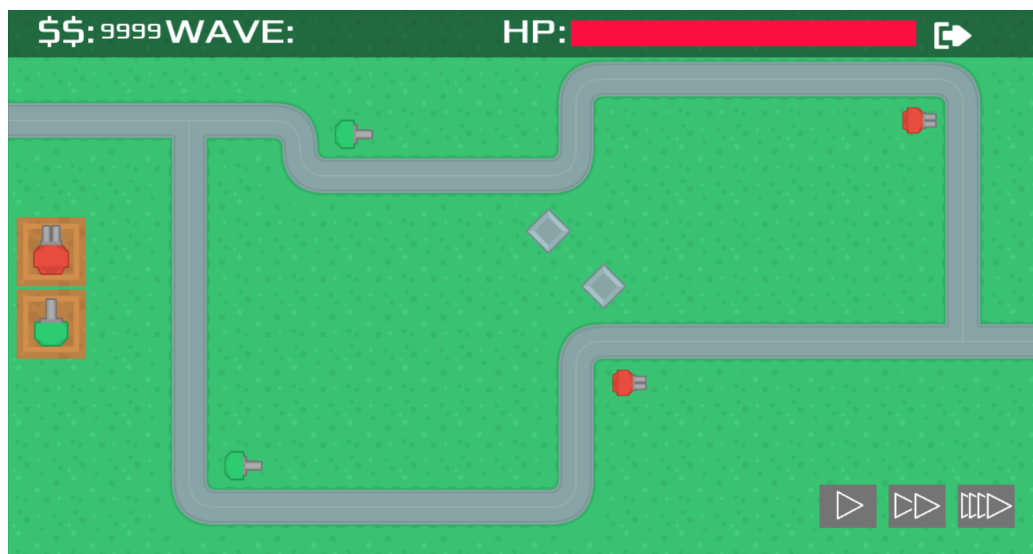


Figura 6.10: *Teste de torres verdes com vermelhas*

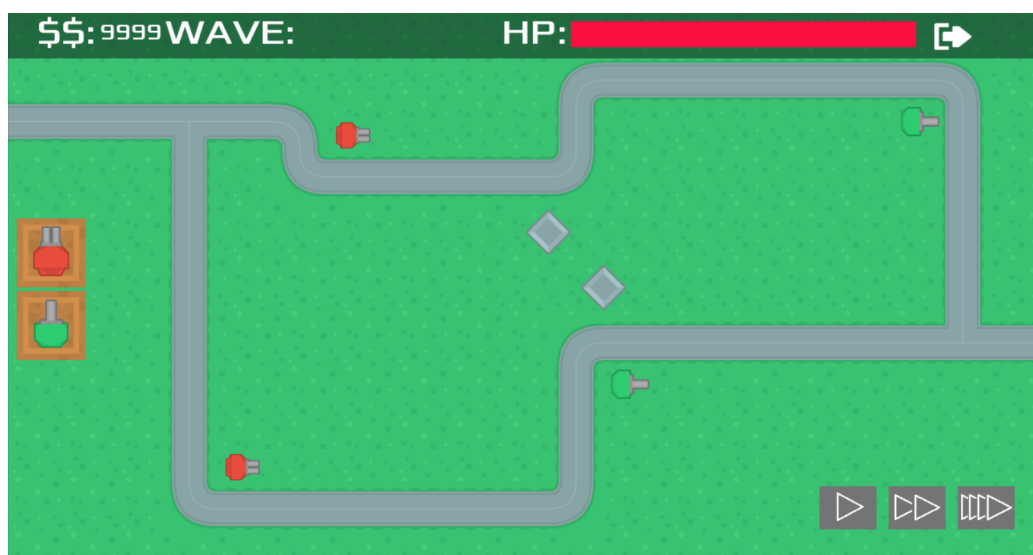


Figura 6.11: *Teste de torres vermelhas com verdes*

6.3.2 Space Shooter

O *Space Shooter* foi configurado de maneira próxima do *Tower Defense*, mas foi preservada a capacidade dos asteroides aparecerem em locais aleatórios entre os 6 possíveis do jogo, exceto quando a IA está gerando os inimigos, onde o algoritmo busca o melhor *spawn*. As naves foram alteradas para efetivamente jogarem a partida; com um nó *Area2D* ao seu redor para detecção e ataque contra o primeiro inimigo que colidir contra o nó, conforme a Figura 6.12; e com a implementação de uma opção de movimentação, visto a Figura 6.13, permitindo testes com uma nave estática no centro da tela, e com movimentação lateral, de lado a lado, pausando nas extremidades.

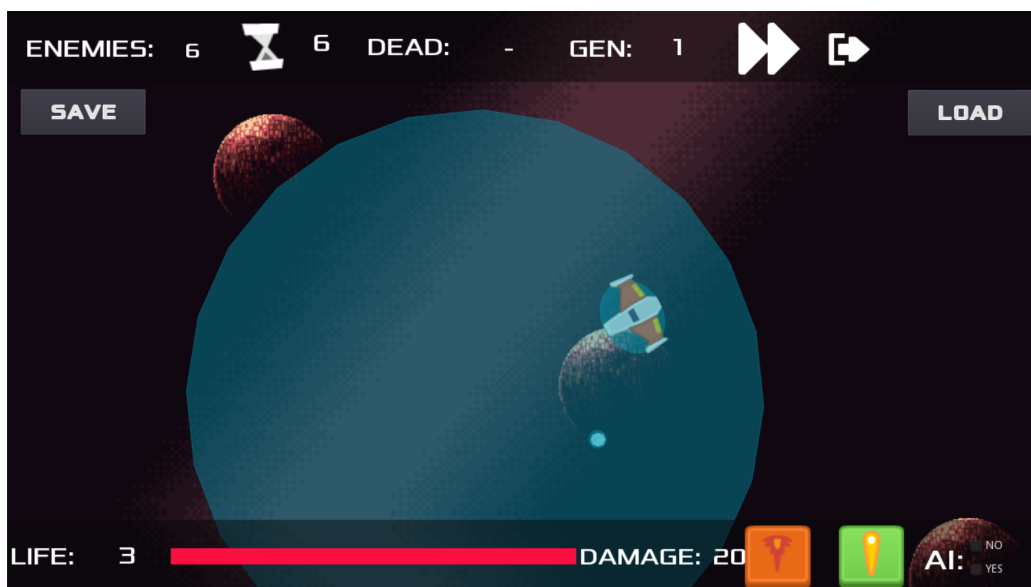


Figura 6.12: Área de detecção da nave em um modo de teste. Imagem retirada do próprio jogo.

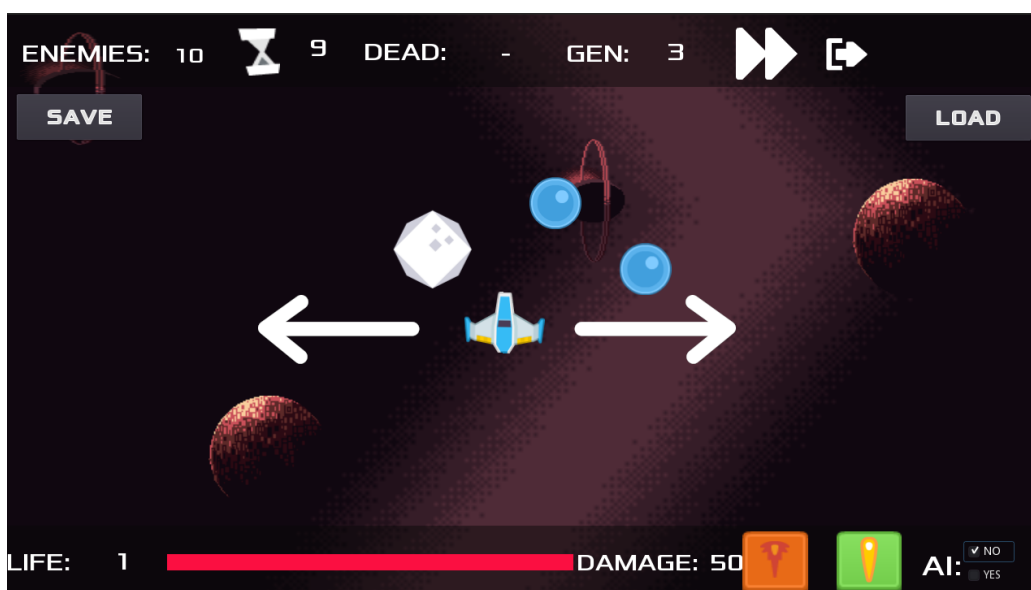


Figura 6.13: Movimentação da nave em um modo de teste. Imagem retirada do jogo e editada.

As opções dos inimigos são semelhantes ao *Tower Defense*, com ondas sempre com os mesmos inimigos, ou um de cada, ou escolhendo aleatoriamente através do *Random-NumberGenerator* da *engine*. Reiterando que nestes casos o *spawn* dos asteroides ocorre em qualquer um dos 6 locais disponíveis na Figura 4.12 escolhidos de maneira aleatória pelo jogo.

Capítulo 7

Apresentação dos Resultados

Para determinar a efetividade do algoritmo genético, foram utilizados métodos estatísticos para comparar o desempenho do mesmo contra outros mecanismos de geração de oponentes. Para aproveitar melhor as características determinísticas do jogo *Tower Defense*, os experimentos se concentraram no mesmo, pois a natureza mais aleatória do *Space Shooter* produziu resultados mais difíceis de analisar e, infelizmente, não havia mais tempo disponível para ajustar os cenários e função *fitness* ou taxa de mutação até encontrar resultados mais conclusivos.

7.1 Organização dos dados

A análise dos dados partiu das estatísticas descritivas de cada onda para todos os experimentos executados, conforme ilustrado pela Tabela 7.1; isto é, foram obtidos 30 amostras de dano para cada i-ésima onda.

	onda 1	onda 2	...	onda n - 1	onda n
experimento 1					
experimento 2	dano	dano		dano	dano
...	da	da	...	da	da
experimento 29	onda	onda		onda	onda
experimento 30					

Tabela 7.1: Forma de coleta dos dados de cada onda.

7.2 Dados das Ondas

As seções seguintes irão dispor os dados considerados relevantes para análise de desempenho do algoritmo, obtidos através da análise exploratória de dados pelos *Jupyter Notebook*¹ desenvolvidos. Dados de ondas sempre com o mesmo tipo de inimigos - chamadas de ondas repetidas - podem mostrar uma possível solução maximal ou próxima dela,

¹ Repositório disponível em <https://github.com/raktanaka/tcc-results> - 24/12/2021

para a qual o algoritmo poderia convergir. Já os dados de ondas com inimigos escolhidos ao acaso - chamadas de ondas aleatórias - indicam um possível valor de dano mínimo, que seria atingido com um método que também não precisa de conhecimento das mecânicas de jogo e de informações sobre os inimigos disponibilizadas.

Para visualização do funcionamento dos algoritmos, foram geradas imagens com as modas² das ondas geradas a partir dos arquivos de texto com os dados. Um *notebook* faz os cálculos necessários e utiliza a biblioteca *Matplotlib* para produzir a imagem com os *sprites* e caminho ou localização do inimigo. As figuras fornecem os inimigos mais comuns em cada onda, mostrando para qual solução o algoritmo tende a convergir. Estas estão disponíveis nos apêndices A, B, C, D, E.

7.2.1 Tower Defense - Ondas Repetidas

Seguem os dados obtidos nos experimentos feito no jogo, para facilidade de leitura, uma cópia da tabela 4.2 da seção 4.1 está abaixo.

	velocidade	dano
tank blue	55	55
tank green	70	45
tank red	80	15
tank orange	120	5
tank purple	90	15
tank yellow	150	5

Tabela 7.2: Dano de cada tanque no Tower Defense

Através da Tabela 7.2 é possível calcular o dano máximo possível como produto dos 12 inimigos que podem ser gerados por *wave* e do maior dano disponível:

$$12 * 55 = 660$$

Numa onda composta inteiramente por tanques verdes (*EnemyGreen*), sem nenhuma eliminação. As Tabelas 7.3, 7.4, 7.5 e 7.6 a seguir mostram os dados de dano médio para cada onda e cada tipo de inimigo, obtidos através das simulações do jogo.

Torres	Inimigos	Dano Médio	Desvio Padrão
Verdes	EnemyGreen	450.00	0.0
Verdes	OneEach	122.35	4.52
Verdes	EnemyPurple	90.06	0.91
Verdes	EnemyRed	89.90	1.22
Verdes	EnemyBlue	60.68	16.77
Verdes	EnemyYellow	40.07	0.57
Verdes	EnemyOrange	39.93	0.57

Tabela 7.3: Desempenho das **ondas repetidas** contra Torres **exclusivamente Verdes**

² Valores mais frequentes em um conjunto de dados

Conforme os dados gerados para torres exclusivamente verdes, nota-se que tanques verdes possuem maior facilidade em causar dano ao jogador, uma vez que possuem a mecânica de **Resistência** contra torres da mesma cor que eles. Portanto, no algoritmo genético, espera-se que esses sejam os indivíduos mais numerosos da população final.

Torres	Inimigos	Dano Médio	Desvio Padrão
Vermelhas	EnemyGreen	360.00	0.0
Vermelhas	EnemyBlue	355.85	28.58
Vermelhas	EnemyRed	180.00	0.0
Vermelhas	OneEach	167.82	2.48
Vermelhas	EnemyPurple	127.60	8.54
Vermelhas	EnemyYellow	50.58	1.61
Vermelhas	EnemyOrange	50.00	0.0

Tabela 7.4: Desempenho das *ondas repetidas* contra Torres *exclusivamente Vermelhas*

Os dados gerados para torres exclusivamente vermelhas apresenta que, em média, tanques Verdes e Azuis são os melhores causadores de dano por onda, mesmo com a resistência dos tanques vermelhos sobre as torres vermelhas. Essa disparidade ocorre pela quantidade de dano que os tanques causam ao chegar no inimigo, conforme Tabela 7.2, o tanque vermelho causa 15 de dano, ou seja, os 12 ($\frac{180}{15}$) tanques sobrevivem ao final de todas as ondas. Contudo, os inimigos verdes e azuis causam mais dano, mesmo que cheguem menos tanques ao final do trajeto, 8 ($\frac{360}{45}$) para tanques verdes e entre 6-7 ($\frac{355}{55}$) nos azuis.

Torres	Inimigos	Dano Médio	Desvio Padrão
Verde + Vermelha	EnemyGreen	420.20	13.91
Verde + Vermelha	EnemyBlue	220.18	3.18
Verde + Vermelha	EnemyRed	148.50	4.50
Verde + Vermelha	OneEach	148.45	4.57
Verde + Vermelha	EnemyPurple	120.00	0.0
Verde + Vermelha	EnemyYellow	50.00	0.0
Verde + Vermelha	EnemyOrange	44.50	1.50

Tabela 7.5: Desempenho das *ondas repetidas* contra 1 Torre Verde e 1 Vermelha (Nessa ordem)

Conforme as informações da Tabela 7.5, percebe-se que o inimigo Verde tem o melhor desempenho de dano por onda repetida, mesmo que sobrevivam menos indivíduos ao final em comparação ao inimigo Vermelho, uma vez que o dano de Verde é 3 vezes maior. Chegam ao final, aproximadamente 9 ($\frac{420}{45}$) Verdes e 10 Vermelhos ($\frac{148.5}{15}$).

Torres	Inimigos	Dano Médio	Desvio Padrão
Vermelha + Verde	EnemyGreen	449.85	2.60
Vermelha + Verde	EnemyBlue	329.82	3.18
Vermelha + Verde	EnemyRed	149.90	1.73
Vermelha + Verde	OneEach	135.05	0.87
Vermelha + Verde	EnemyPurple	120.00	0.0
Vermelha + Verde	EnemyYellow	50.00	0.0
Vermelha + Verde	EnemyOrange	49.45	1.57

Tabela 7.6: Desempenho das ondas repetidas contra 1 Torre Vermelha e 1 Verde(Nessa ordem)

Para a Tabela 7.6, nota-se um aumento de desempenho significativo do inimigo Azul e Verde, onde sobrevive um tanque a mais verde e quase 2 azuis em relação ao anterior. Contudo, mantém-se a dominância do dano médio por onda do inimigo Verde. Os desvios padrões baixos - o mais alto apresentado nas Torres Verdes contra *EnemyBlue* é de aproximadamente 25% do dano total, mas é uma exceção considerando todos os resultados - mostram a consistência do jogo.

Conforme os dados gerados, o inimigo Verde indica ter o melhor desempenho de dano nas ondas repetidas, contudo, mais para frente desse capítulo, será mostrado que eles não serão os mais selecionados, por conta da função *fitness* escolhida, que prioriza sobrevivência ao invés de dano por wave.

7.2.2 Tower Defense - Ondas Aleatórias

Foram calculadas as médias de dano causadas por cada onda aleatória de distribuição uniforme (Tabela 7.7) e o máximo de dano causado em qualquer onda (Tabela 7.8), considerando todos os experimentos. A Figura 7.1 ilustra a onda que causou o maior dano.

Torres	Dano Médio	Desvio Padrão
Verdes	81.42	55.00
Vermelhas	125.35	56.98
Verde + Vermelha	100.37	54.16
Vermelha + Verde	102.37	56.62

Tabela 7.7: Média e Desvio Padrão do dano em ondas aleatórias no Tower Defense.

Torres	Onda	Dano Total
Verdes	[EnemyGreen, 1]; [EnemyGreen, 1]; [EnemyGreen, 0]; [EnemyRed, 1]; [EnemyPurple, 0]; [EnemyGreen, 0]; [EnemyRed, 1]; [EnemyGreen, 0]; [EnemyOrange, 0]; [EnemyGreen, 0]; [EnemyGreen, 0]; [EnemyRed, 0]	290
Vermelhas	[EnemyBlue, 0]; [EnemyRed, 0]; [EnemyBlue, 0]; [EnemyGreen, 0]; [EnemyGreen, 1]; [EnemyRed, 1]; [EnemyBlue, 0]; [EnemyOrange, 1]; [EnemyBlue, 0]; [EnemyYellow, 0]; [EnemyOrange, 0]; [EnemyYellow, 1]	355
Verde + Vermelha	[EnemyPurple, 0]; [EnemyBlue, 0]; [EnemyBlue, 0]; [EnemyBlue, 1]; [EnemyRed, 0]; [EnemyGreen, 1]; [EnemyYellow, 0]; [EnemyRed, 1]; [EnemyGreen, 1]; [EnemyGreen, 0]; [EnemyGreen, 0]; [EnemyYellow, 1]	330
Vermelha + Verde	[EnemyYellow, 0]; [EnemyBlue, 0]; [EnemyRed, 1]; [EnemyGreen, 0]; [EnemyGreen, 1]; [EnemyRed, 1]; [EnemyGreen, 0]; [EnemyYellow, 0]; [EnemyRed, 1]; [EnemyGreen, 0]; [EnemyGreen, 0]; [EnemyPurple, 1]	315

Tabela 7.8: Ondas aleatórias com maior dano no Tower Defense.

Torres	Inimigos	Dano
Verdes	 S  S  N  S  N  N  S  N  N  N  N  N	290
Vermelhas	 N  N  N  N  S  S  N  S  N  N  N  S	355
Verdes + Vermelhas	 N  N  N  S  N  S  N  S  S  N  N  S	330
Vermelhas + Verdes	 N  N  S  N  S  S  N  N  S  N  N  S	315

Figura 7.1: Visualização das ondas aleatórias com maior dano no Tower Defense.

7.2.3 Space Shooter - Ondas Repetidas

Foram executados experimentos análogos ao Tower Defense no Space Shooter, segue uma cópia da tabela 4.4 da seção 4.2 para facilitar leitura.

	Velocidade	Dano	Vida
inimigos	500	20	50
inimigo1	350	45	30
inimigo2	470	50	40
inimigo3	320	60	40
inimigo4	550	20	60
inimigo5	700	10	120

Tabela 7.9: Velocidade, dano e vida de cada asteroide no Space Shooter

Através da Tabela 7.9, é possível calcular o dano máximo possível como a onda de seis inimigos com o maior dano disponível:

$$6 * 60 = 360$$

Numa onda composta inteiramente pelo *inimigo3*, sem nenhuma eliminação. As tabelas 7.10, 7.11, 7.12 e 7.13 a seguir mostram os dados de dano médio para cada onda e cada tipo de inimigo.

Nave	Inimigo	Dano Médio	Desvio Padrão
Disparo Amarelo - Parada	Inimigo3	314.93	42.46
Disparo Amarelo - Parada	Inimigo2	299.78	3.33
Disparo Amarelo - Parada	Inimigo1	181.50	41.67
Disparo Amarelo - Parada	Inimigo4	120.00	0.0
Disparo Amarelo - Parada	Inimigos	120.00	0.0
Disparo Amarelo - Parada	OneEach	82.75	27.47
Disparo Amarelo - Parada	Inimigo5	60.00	0.0

Tabela 7.10: Ondas repetidas contra IA de Disparo Amarelo e Parado

Na tabela, nota-se que os valores piores classificados são as ondas em que todos os asteroides chegaram no Jogador, mas possuem danos muito baixos ($\frac{\text{dano médio}}{\text{dano individual do asteróide}}$), enquanto os que possuem danos melhores perderam algum indivíduo da população no caminho, pois possuem valores menores que os danos totais possíveis (dano * 6, onde 6 é quantidade de inimigos) e seu desvio padrão é alto.

Nave	Inimigo	Dano Médio	Desvio Padrão
Disparo Amarelo - Movendo	Inimigo3	213.20	52.43
Disparo Amarelo - Movendo	Inimigo2	184.61	41.98
Disparo Amarelo - Movendo	Inimigo1	130.10	48.15
Disparo Amarelo - Movendo	OneEach	100.85	43.75
Disparo Amarelo - Movendo	Inimigo4	80.69	15.70
Disparo Amarelo - Movendo	Inimigos	67.69	15.46
Disparo Amarelo - Movendo	Inimigo5	39.23	7.71

Tabela 7.11: Ondas repetidas contra IA de Disparo Amarelo e Movendo

Para esses dados, nota-se que, com a Nave se movendo, a chance dos inimigos acertarem o Jogador decai bastante em todos os casos, contudo, os tipos de inimigos mais promissores ainda se mantêm como Inimigo3, Inimigo2 e Inimigo1.

Nave	Inimigo	Dano Médio	Desvio Padrão
Disparo Vermelho - Parada	Inimigo3	241.60	56.01
Disparo Vermelho - Parada	Inimigo2	240.94	56.09
Disparo Vermelho - Parada	OneEach	125.99	38.03
Disparo Vermelho - Parada	Inimigo4	106.00	21.81
Disparo Vermelho - Parada	Inimigos	102.44	20.04
Disparo Vermelho - Parada	Inimigo5	62.57	17.74
Disparo Vermelho - Parada	Inimigo1	27.60	29.54

Tabela 7.12: Ondas repetidas contra IA de Disparo Vermelho e Parado

Para esses dados, percebe-se que o Disparo Vermelho tem uma vantagem muito maior contra **Inimigo1**, um dos candidatos para o indivíduo mais apto da população nos testes para o Disparo Amarelo, principalmente por matá-lo em um único disparo.

Nave	Inimigo	Dano Médio	Desvio Padrão
Disparo Vermelho - Movendo	Inimigo3	169.87	63.54
Disparo Vermelho - Movendo	Inimigo2	169.67	49.77
Disparo Vermelho - Movendo	OneEach	96.99	39.72
Disparo Vermelho - Movendo	Inimigo4	76.29	18.05
Disparo Vermelho - Movendo	Inimigos	65.89	18.23
Disparo Vermelho - Movendo	Inimigo5	39.26	7.40
Disparo Vermelho - Movendo	Inimigo1	24.60	28.60

Tabela 7.13: Ondas repetidas contra IA de Disparo Vermelho e Movendo

Novamente, a classificação dos indivíduos que parecem mais aptos não muda muito, somente decaiu o Dano Médio por onda já que a Nave agora está se **movendo**, dificultando que os asteroides a acertem.

Essas tabelas das ondas repetidas são importantes pois ajudam a inferir sobre os resultados que levaram aos indivíduos na população final após a execução do algoritmo genético. Também mostram, através dos desvios padrões altos - diversos testes mostram valores que chegam a ser maiores que o dano médio - bastante acima dos obtidos no *Tower Defense* que a variabilidade de cada onda é alta, e irá interferir nos resultados.

7.2.4 Space Shooter - Ondas Aleatórias

Foram calculadas as médias de dano causado por cada onda aleatória (Tabela 7.14) e o máximo de dano causado em qualquer onda (Tabela 7.15), considerando todos os experimentos. A Figura 7.2 ilustra a onda que causou o maior dano, um resultado importante para embasar que o Algoritmo Genético possui um desempenho melhor do que Ondas de Distribuição Uniforme Aleatórias.

Nave	Dano Médio	Desvio Padrão
Disparo Amarelo - Parada	181.26	46.18
Disparo Amarelo - Movendo	124.95	47.32
Disparo Vermelho - Parada	123.66	50.84
Disparo Vermelho - Movendo	91.77	46.11

Tabela 7.14: Média e Desvio Padrão do dano em ondas aleatórias no Space Shooter.

Nave	Onda	Dano Total
Disparo Amarelo - Parada	[inimigo1, (-100, 300)]; [inimigo3, (-100, 100)]; [inimigo1, (-100, 300)]; [inimigo1, (-100, 300)]; [inimigos, (90, -50)]; [inimigo3, (-100, 100)]	275
Disparo Amarelo - Movendo	[inimigo3, (-100, 100)]; [inimigo1, (-100, 300)]; [inimigo3, (-100, 100)]; [inimigo1, (-100, 300)]; [inimigo1, (-100, 300)]; [inimigo3, (-100, 100)]	315
Disparo Vermelho - Parada	[inimigo3, (-100, 100)]; [inimigo3, (-100, 100)]; [inimigo5, (1350, 500)]; [inimigo4, (200, -50)]; [inimigo3, (-100, 100)]; [inimigo2, (1350, 100)]	260
Disparo Vermelho - Movendo	[inimigo3, (90, -50)]; [inimigo1, (90, -50)]; [inimigo3, (90, -50)]; [inimigo1, (90, -50)]; [inimigo5, (90, -50)]; [inimigo1, (90, -50)]	350

Tabela 7.15: Ondas aleatórias com maior dano no Space Shooter.

Nave	Onda												Dano
Disparo Amarelo Parada		6		5		6		5		5		6	315
Disparo Amarelo Movendo		5		6		5		5		1		6	275
Disparo Vermelho Parada		1		1		1		1		1		1	350
Disparo Vermelho Movendo		6		6		4		2		6		3	260

Figura 7.2: Visualização das ondas aleatórias com maior dano no Space Shooter.

7.3 Fitness dos Jogos

7.3.1 Fitness e Mutação

Foram testados três modelos de *fitness* no algoritmo genético para o jogo *Tower Defense*, buscando obter melhores resultados na geração de ondas; enquanto o jogo *Space Shooter* foi testado somente com a primeira e a terceira versão. Foi adotada uma nomenclatura de versões para simplificar as referências no texto:

	<i>Fitness</i> (v1)	<i>Fitness</i> (v2)
Taxa de mutação (v1)	TD e SS	-
Taxa de mutação (v2)	-	TD
Taxa de mutação (v3)	SS	TD

Tabela 7.16: Relação dos *fitness* e taxa de mutação usados no TD e SS.

7.3.2 Tower Defense - Fitness e Mutação

O primeiro teste realizado utilizou uma função *fitness* que considera se o alvo chegou ao final do trajeto e o quanto do trajeto foi percorrido, indivíduos que completaram o percurso serão muito mais recompensados e aqueles que morreram muito longe do alvo tendem a ser eliminados da população. A taxa de mutação decai conforme o tempo, começando em *mutation_prob* = 1.0 (100%) e decrementando até 0. A taxa de mutação volta para 100% a cada repetição do experimento (30 ondas).

Programa 7.1 *Fitness* do primeiro teste do TD (v1).

```

1  FUNCAO fitness_v1 (x)  ▷ Dá uma pontuação para indivíduo x
2  ▷ reached_goal(x) = booleana se chegou ou não ao final do trajeto
3      se reached_goal(x)
4          fit ← 1
5      senao
6          fit ← 0
7
8  ▷ offset(x) = quanto do percurso foi completado.
9      fit ← (fit + offset(x)) / 2  ▷ média aritmética dos valores de x
10     devolva fit  ▷ Um float com uma pontuação no intervalo [0,1].
11 fim
```

Programa 7.2 Taxa de mutação do TD para o primeiro teste (v1).

```

1  FUNCAO mutation_v1 ()
2      se mutation_prob ≥ 0
3          mutation_prob -= 0.05
4      ...
5  ▷ continuação do processo de mutação.
6  fim
```

Para o segundo teste, a função *fitness* considera a quantidade do trajeto que foi percorrido (*float* de 0 a 1) e a quantidade de vida ao final do trajeto (*float* de 0 a 1), ou seja, inimigos que completarem o percurso serão mais recompensados conforme a quantidade de vida que chegaram ao final do trajeto. Note que indivíduos que morreram no caminho não serão bem recompensados com essa função. Por um erro, a taxa de mutação ficou em 1 a partir da segunda wave, contudo, alguns resultados interessantes foram obtidos também.

Programa 7.3 *Fitness do TD (v2) (Avaliação, em português).*

```

1  FUNCAO fitness_TD_v2 (x)  ▷ Dá uma pontuação para indivíduo x
2  ▷ offset(x) = quanto do trajeto foi percorrido, float de 0 a 1.
3  ▷ hp (x) = quanto de HP sobrou do inimigo, float de 0 a 1.
4  fit ← (offset(x) + hp(x)) / 2  ▷ média aritmética dos valores de x
5  devolva fit  ▷ Pontuação no intervalo [0,1].
6  fim

```

Programa 7.4 Taxa de mutação do TD para cada bateria de testes (2º par dos gráficos).

```

1  FUNCAO mutation_v2 ()
2  se geração = 1
3  mutation_prob ← 1 / 12
4
5  senão
6  mutation_prob ← 1.0
7  ...
8  ▷ continuação do processo de mutação.
9  fim

```

Para o terceiro teste, a função *fitness* considera a quantidade do trajeto que foi percorrido (*float* de 0 a 1) e a quantidade de vida ao final do trajeto (*float* de 0 a 1), idêntico ao anterior. Contudo, a taxa de mutação permanece em um valor fixo equivalente a $\frac{1}{12}$, conforme embasamento no estudo de [HAUPT \(2000\)](#).

Programa 7.5 *Fitness do TD (v3) (Avaliação, em português).*

```

1  FUNCAO fitness_TD_v3 (x)  ▷ Dá uma pontuação para indivíduo x
2  ▷ offset (x) = quanto do trajeto foi percorrido, float de 0 a 1.
3  ▷ hp (x) = quanto de HP sobrou do inimigo, float de 0 a 1.
4  fit ← ( offset (x) + hp (x)) / 2  ▷ média aritmética dos valores de x
5  devolva fit  ▷ Pontuação no intervalo [0,1].
6  fim

```

Programa 7.6 Taxa de mutação v3 do TD para cada bateria de testes (3º par dos gráficos).

```

1  FUNCAO mutation_v3 ()
2      mutation_prob ← 1 / 12
3      ...
4      ▷ continuação do processo de mutação.
5  fim

```

7.3.3 Space Shooter - Fitness e Mutação

Conforme a Tabela 7.16, temos uma única função *fitness* e as mesmas taxas de mutação 1 e 2 apresentadas na subseção anterior.

Programa 7.7 *Fitness* de todos os testes do SS (v1).

```

1  FUNCAO fitness_v1 (x)  ▷ Dá uma pontuação para indivíduo x
2      ▷ reached_goal(x) = booleana se acertou o inimigo
3      se reached_goal(x)
4          fit ← 5
5      senao
6          fit ← 0
7
8      ▷ hp (x) = 5 * quanto de HP restou ao final da rodada.
9      fit ← (fit + hp (x)) / 10  ▷ média aritmética dos valores de x
10     devolva fit  ▷ Um float com uma pontuação no intervalo [0,1].
11 fim

```

7.4 Avaliação dos Testes de Versões de Algoritmo Genético

Para a avaliação dos algoritmos genéticos testados, foram gerados gráficos com a média de dano de cada *i*-ésimas ondas (de 1 até 30) e uma linha contínua com a regressão polinomial de grau 5 (7.8), para os 30 experimentos em cada versão testada.

Analisando os gráficos obteve-se a onda em que ocorreu a convergência do algoritmo, isto é, o ponto a partir do qual o dano causado pela onda ficou estável e aproximadamente linear.³

³ Através de análise visual das linha de regressão polinomial, procurou-se o primeiro ponto de mínimo ou máximo - dependendo do comportamento crescente ou decrescente do algoritmo - a partir do qual a curva se mantinha razoavelmente estável. Testes que apresentaram resultados constantemente crescentes, decrescentes ou oscilantes foram considerados instáveis, onde o algoritmo nunca convergiu

Programa 7.8 Trecho do código para cálculo da regressão polinomial.

```

1  from numpy.polynomial import Polynomial
2
3  fit = Polynomial.fit(df_1['wave number'], df_1['average damage'], deg=5)

```

A partir dos mesmos dados sem qualquer processamento - isto é, informações de 300 ondas com 6 ou 12 inimigos e o dano causado - foram gerados gráficos de *boxplot*⁴ padrão, com a média representada por quadrados vermelhos. Nestes é possível verificar a dispersão dos danos gerados pelo algoritmo, afim de verificar se o mesmo tende a chegar no mesmo resultado com consistência (indicados no gráfico por "caixas" mais compactas pois mais resultados de ondas se concentrariam de maneira próxima), ou se produz muitos resultados discrepantes, seja com dano muito alto ou muito baixo (graficamente seriam "caixas" longas, ou *outliers*), e onde estão a a maior parte das soluções encontradas (onde a mediana divide os 30 experimentos pela metade e indica onde estão os resultados de dano mais frequente, enquanto a média pode ser influenciada por *outliers*).

7.4.1 Tower Defense - Resultados

Os gráficos seguintes (7.3, 7.4, 7.5 e 7.6) apresentam os resultados dos quatro casos de teste no jogo *Tower Defense*, para as três opções citadas em 7.3.1.

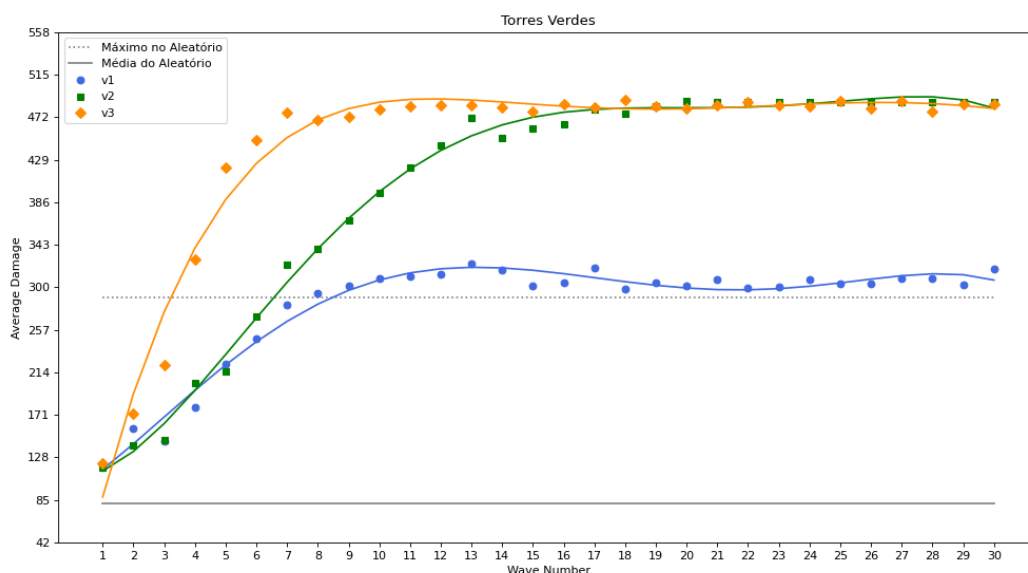


Figura 7.3: Gráfico com as médias de dano para cada onda no teste com as Torres Verdes para as versões v1, v2 e v3.

⁴ Representação gráfica de localidade, viés e dispersão através dos quartis

7.4 | AVALIAÇÃO DOS TESTES DE VERSÕES DE ALGORITMO GENÉTICO

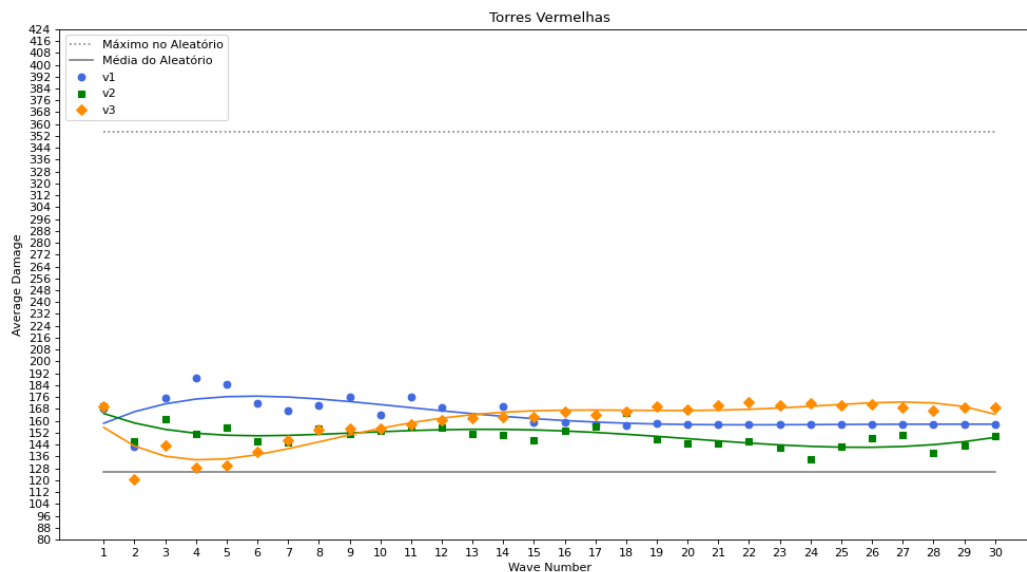


Figura 7.4: Gráfico com as médias de dano para cada onda no teste com as Torres Vermelhas para as versões v1, v2 e v3.

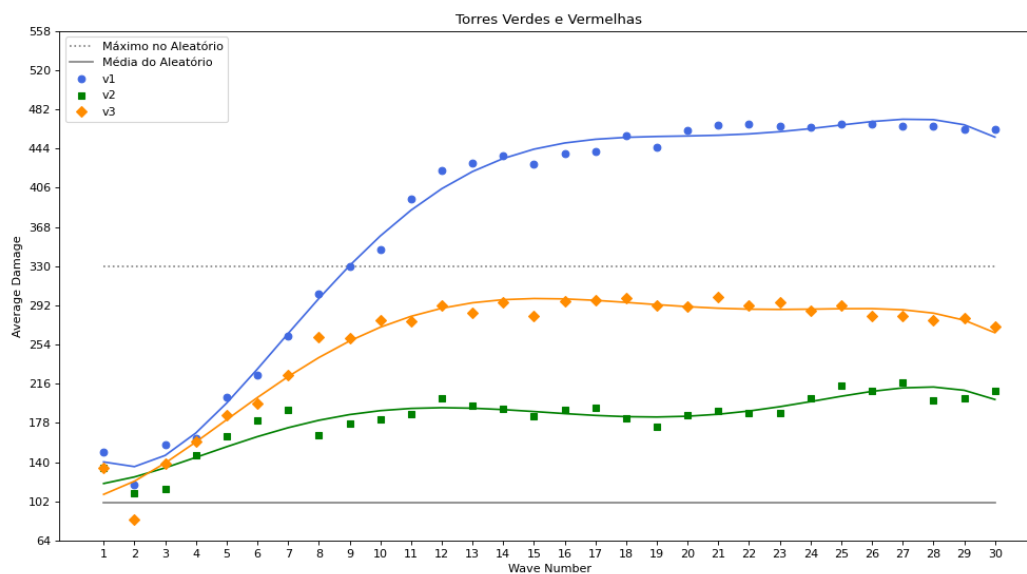


Figura 7.5: Gráfico com as médias de dano para cada onda no teste com as Torres Verdes + Vermelhas para as versões v1, v2 e v3.

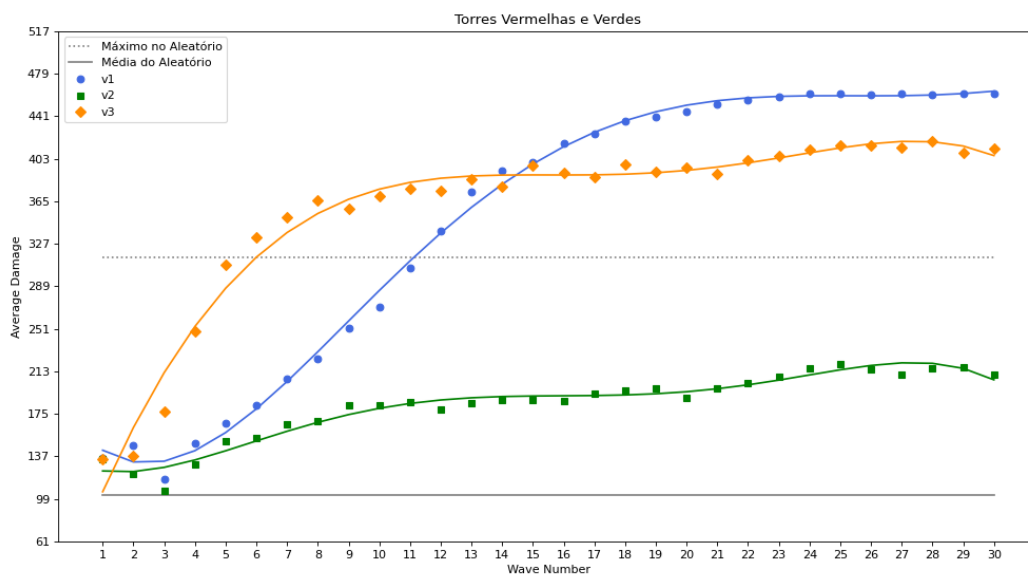


Figura 7.6: Gráfico com as médias de dano para cada onda no teste com as Torres Vermelhas + Verdes para as versões v1, v2 e v3.

As Figuras 7.7, 7.8, 7.9 e 7.10 a seguir mostram o comportamento, em cada teste, do dano total de cada i -ésima onda, cada uma sendo representada num *boxplot*.

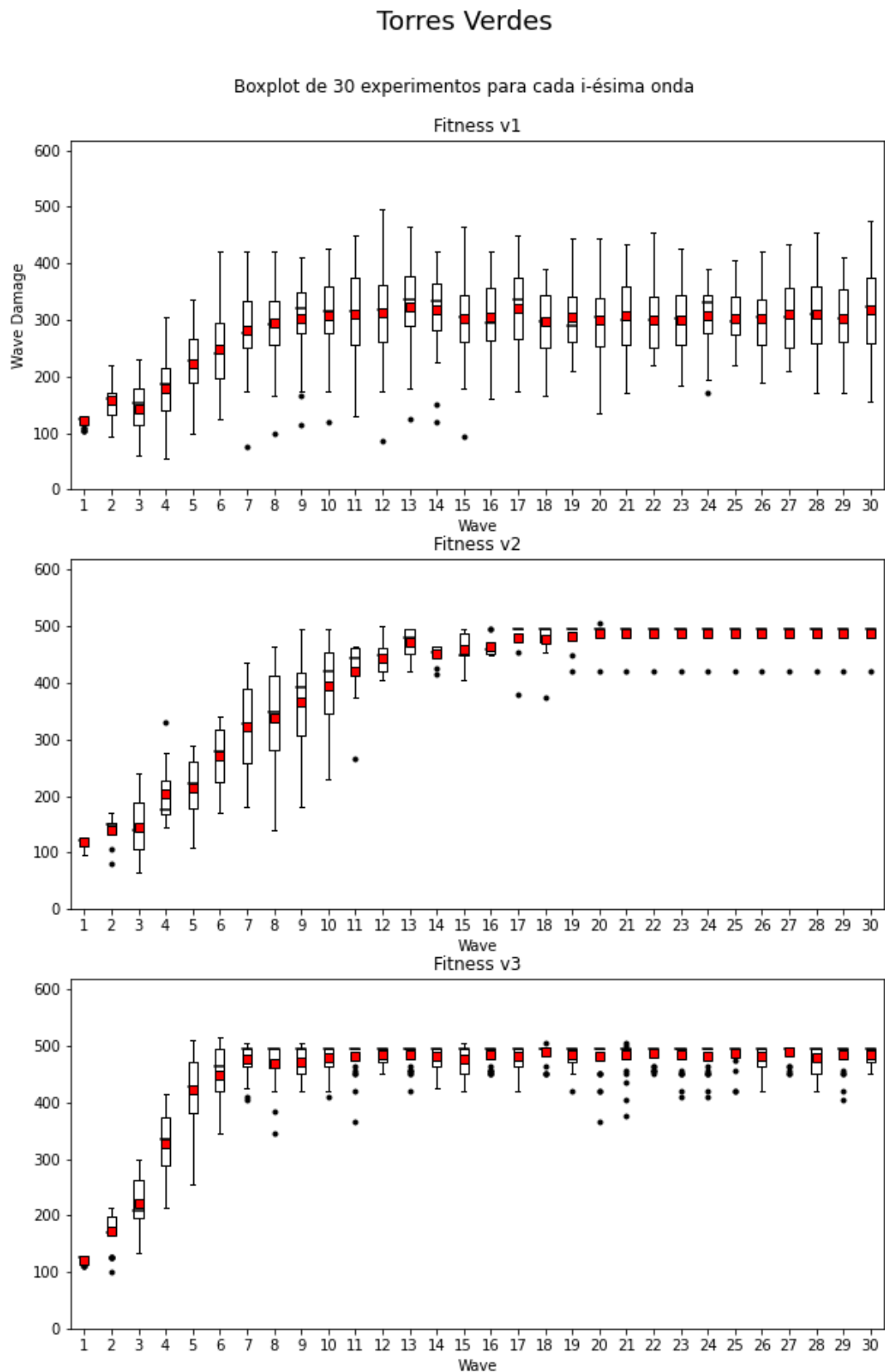


Figura 7.7: Boxplot do dano das 30 ondas nos 30 experimentos, para as versões v1, v2 e v3.

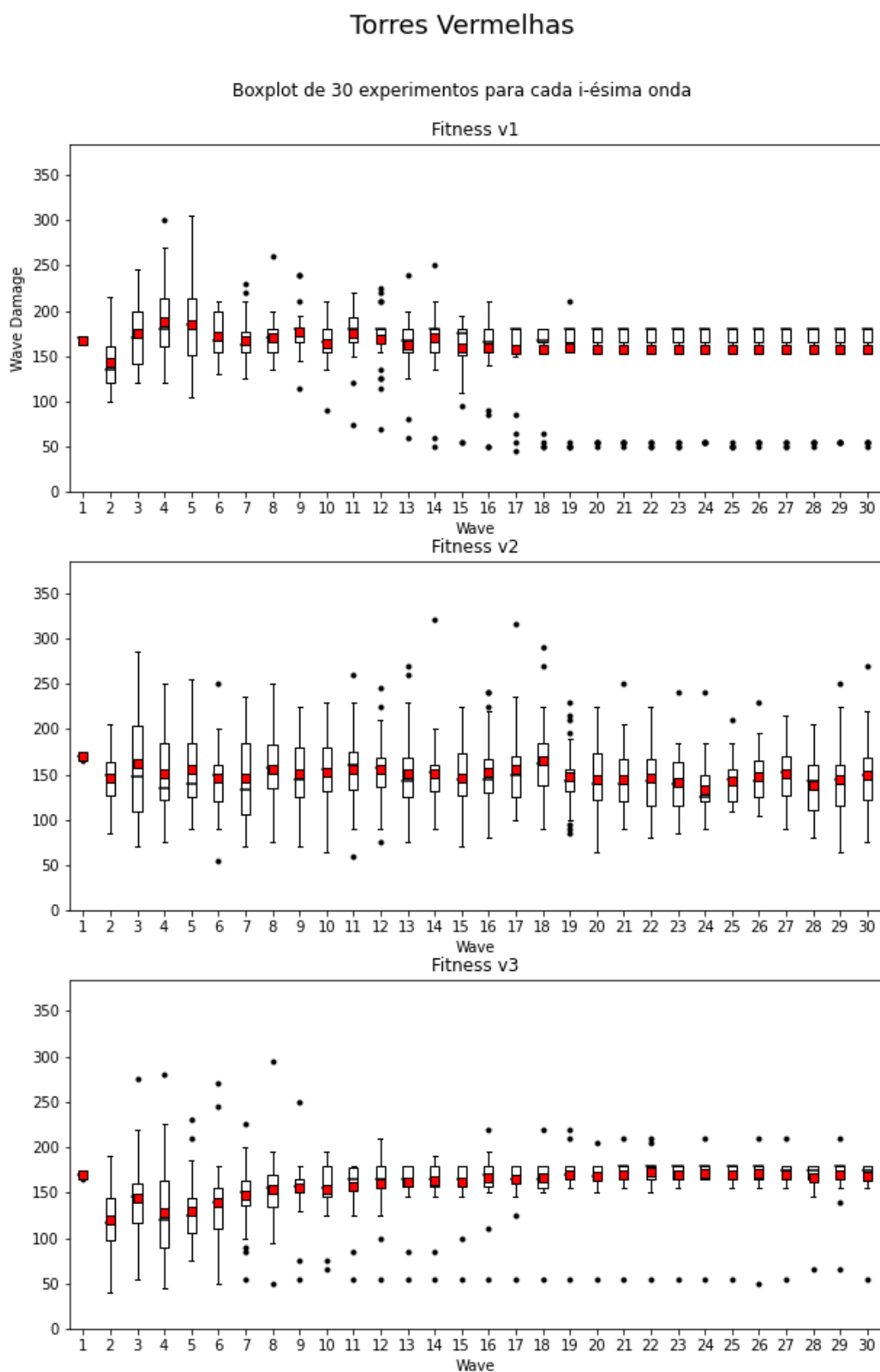


Figura 7.8: Boxplot do dano das 30 ondas nos 30 experimentos, para as versões v1, v2 e v3.

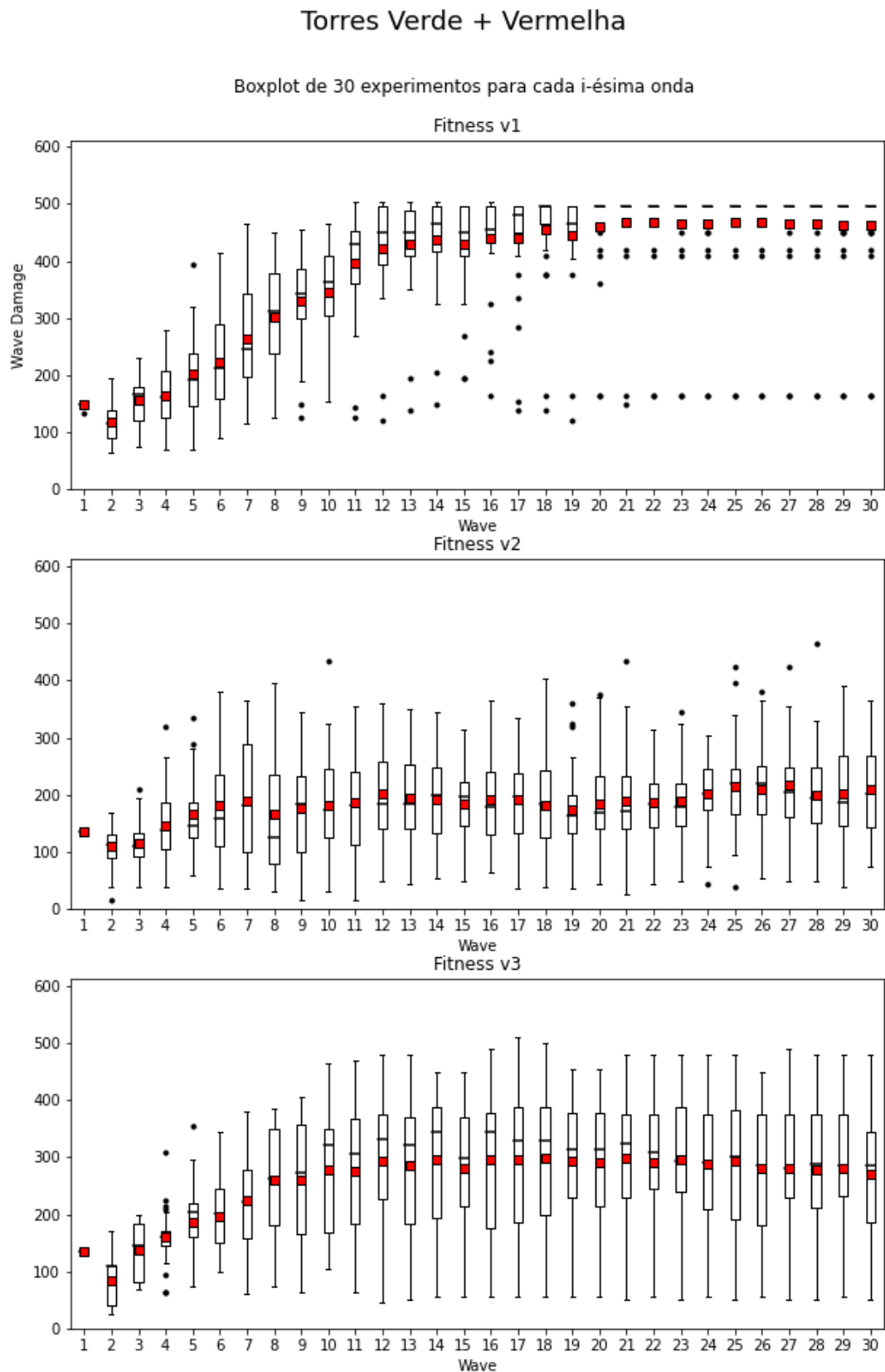


Figura 7.9: Boxplot do dano das 30 ondas nos 30 experimentos, para as versões v1, v2 e v3.

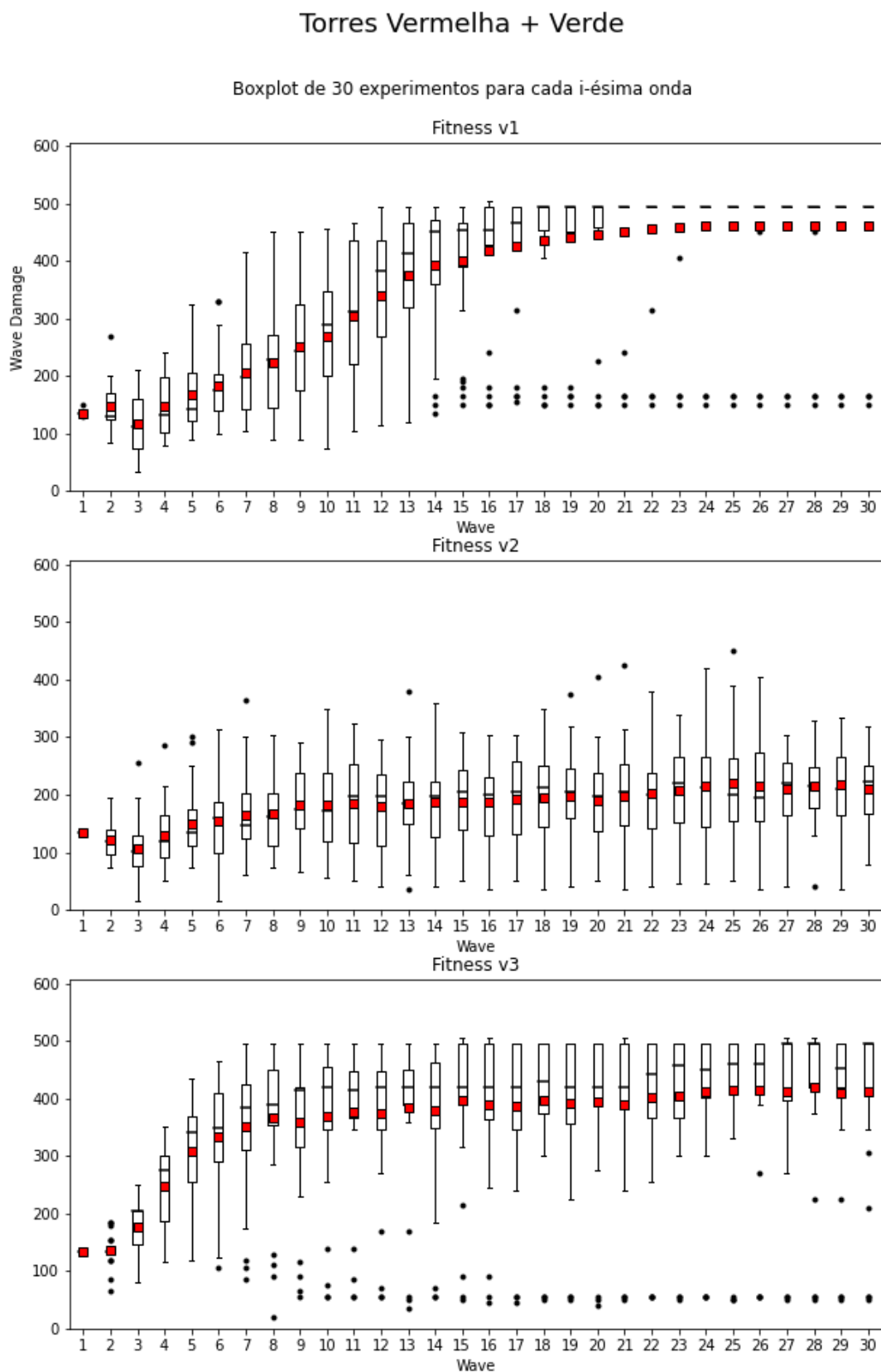


Figura 7.10: Boxplot do dano das 30 ondas nos 30 experimentos, para as versões v1, v2 e v3.

A Tabela 7.17 mostra a onda aproximada onde o algoritmo parece ter convergido, e o dano médio após a estabilidade:

Torres	v1		v2		v3	
	Convergiu	Dano Médio	Convergiu	Dano Médio	Convergiu	Dano Médio
Verdes	8	307.04	12	478.61	8	482.57
Vermelha	15	157.78	Não	-	12	167.44
Verde + Vermelha	18	463.26	Não	-	11	288.18
Vermelha + Verde	21	459.30	Não	-	12	399.16

Tabela 7.17: Tabela mostrando a onda onde ocorreu a convergência e o dano médio a partir desse ponto até o final.

7.4.2 Tower Defense - Comparação

No *Tower Defense* todas as versões do Algoritmos Genéticos se mostraram adequadas, superando a média da geração aleatória, entretanto no teste com Torres Vermelhas nenhuma superou o maior dano obtido em uma onda do teste *Random*. Contra Torres Verdes existe uma solução ótima óbvia, tanques Verdes (*EnemyGreen*), onde o inimigo com maior resistência (e portanto vida) é o mesmo com o maior dano, e as três versões testadas parecem chegar nesse resultado.

Individualmente, as seguintes características foram apresentadas:

v1:

- Convergência em todos os testes;
- Contra Torres Verdes apresenta a maior dispersão comparado com v2 e v3, mas o maior dano médio. Único caso onde a média e mediana ficaram próximas;
- Menor dispersão nos outros testes, após convergência;
- Exceto contra Torres Verdes, mostra dispersão inicial alta, quando converge concentra o dano no último quartil (Q4) e em *outliers* próximos dos menores valores possíveis nas ondas repetidas, fornecidos na Tabela 7.2.1, mas com mediana acima da média;
- Não converge para um estado maximal contra Torres Vermelhas, mas consegue superar o maior dano aleatório (pois o melhor estado maximal detectado pela função *fitness* é priorizar sobrevivência dos tanques ao invés do dano causado no Jogador);
- Maior dano, dentre as versões, contra Torres heterogêneas (Verde e Vermelha, Vermelha e Verde);

v2:

- Somente converge contra Torres Verdes - onde apresenta o segundo melhor resultado - concentrando resultados na mediana e em *outliers* ligeiramente abaixo;
- Nos outros casos estabiliza de maneira oscilante, sem melhorar significativamente em relação a onda inicial, com poucos *outliers* mas muita dispersão (devido à taxa de mutação ser 100% após a primeira onda);
- Consistente, com médias e medianas próximas.

v3:

- Convergência em todos os testes, e mais rápida do que v1;
- Melhor resultado médio contra Torres Verdes - acima do maior dano aleatório - com dados concentrados no segundo e terceiro quartil, mediana acima da média e alguns *outliers* ligeiramente abaixo;
- No teste contra Torres Vermelhas apresenta concentração de resultados entre a mediana no último quartil (Q4) e *outliers* próximos das médias das piores ondas aleatórias;
- Apresenta a maior dispersão das versões do algoritmo contra Torres Verde + Vermelha, os quartis se estendendo desde o melhor resultado do v1 até os piores apresentados pelas ondas repetidas (Tabela 7.2.1);
- Contra Torres Vermelha + Verde também supera o maior dano aleatório, com resultados concentrados entre a medianas no último quartil (Q4) e média, que foi afetada por *outliers*.

Os resultados indicam que a versão v1 é bastante consistente, com baixa dispersão em 3 testes, mas corre risco de convergir em resultados longe do ótimo, possivelmente por progressivamente reduzir os genes aos piores candidatos e estabilizar no menos pior (taxa de mutação zero após a 20ª onda).

A última versão testada (v3) mostrou comportamento semelhante ao v1 mas com danos menores, exceto contra Torres Verdes. No restante não foi capaz de eliminar casos extremos onde estabilizou próximo do pior caso, apesar da pequena melhoria contra Torres Verde + Vermelha, provavelmente por reduzir os candidatos aptos por más escolhas, semelhante a v1.

7.4.3 Space Shooter - Resultados

Os gráficos seguintes (7.11, 7.12, 7.13 e 7.14) apresentam os resultados dos quatro casos de teste no jogo *Space Shooter*, para as três opções citadas em 7.3.1.

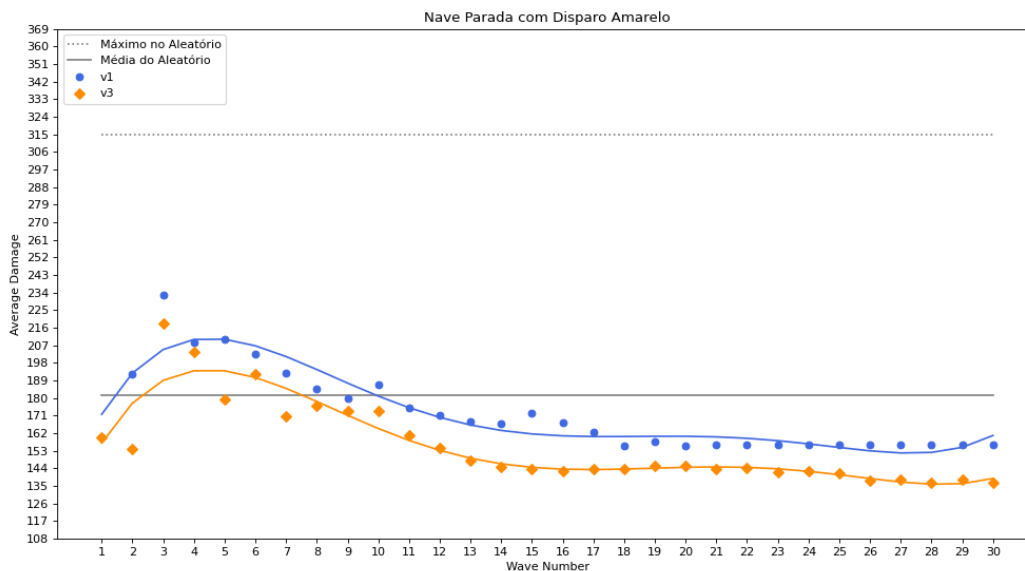


Figura 7.11: Gráfico com as médias de dano para cada onda no teste com a Nave Parada, Disparo Amarelo para as versões v1 e v3.

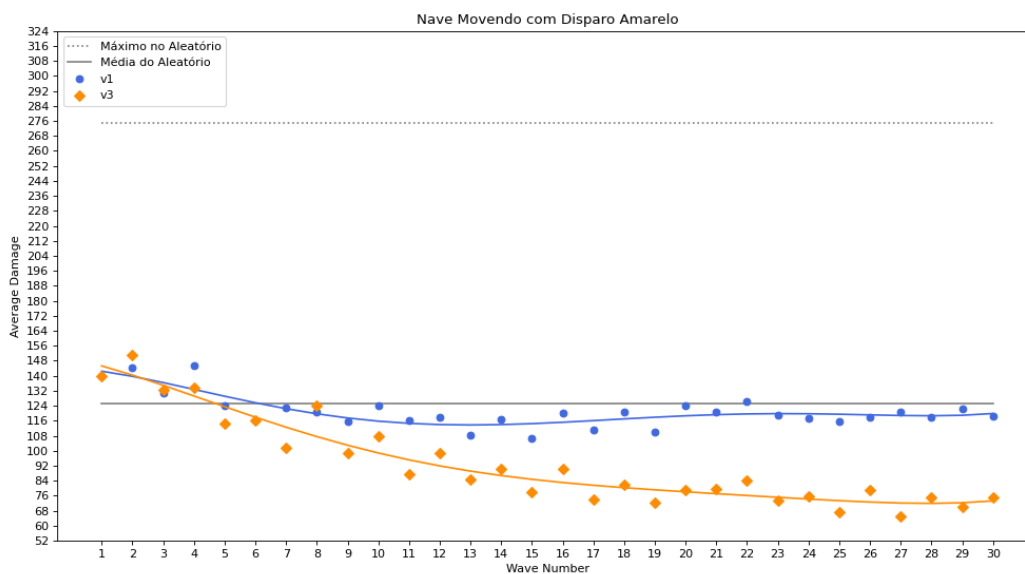


Figura 7.12: Gráfico com as médias de dano para cada onda no teste com a Nave Movendo, Disparo Amarelo para as versões v1 e v3.

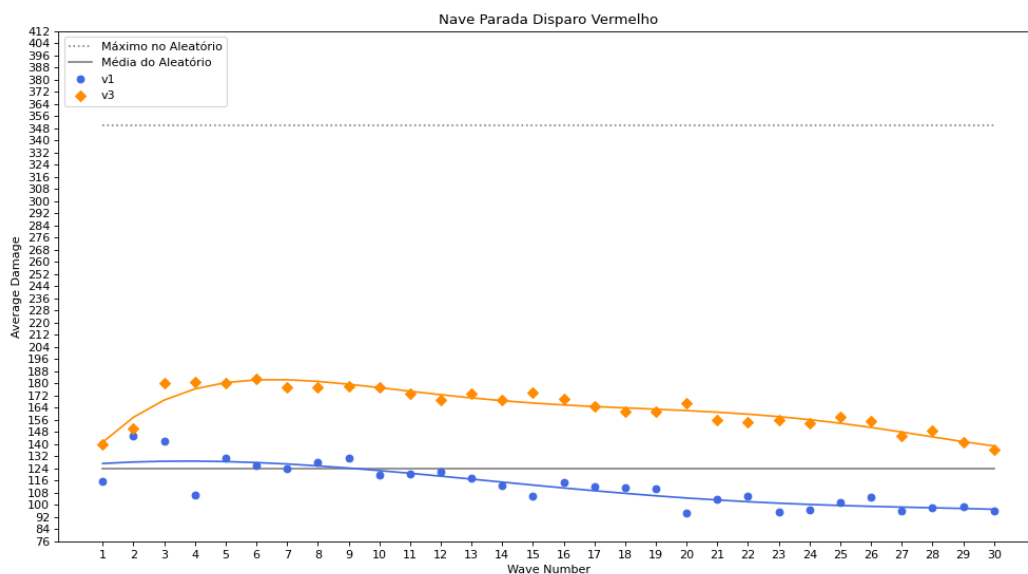


Figura 7.13: Gráfico com as médias de dano para cada onda no teste com a Nave Parada, Disparo Vermelho para as versões v1 e v3.

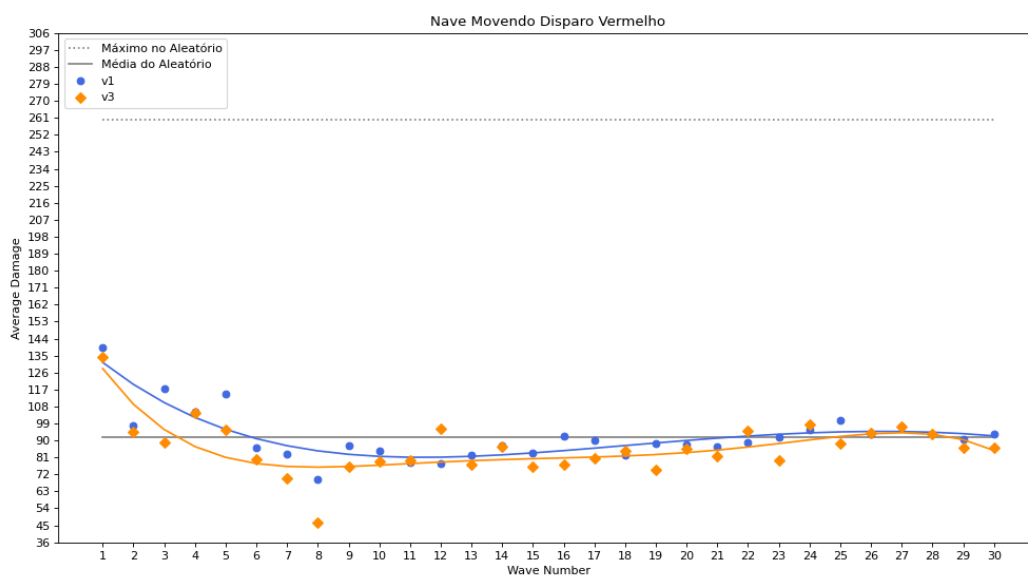


Figura 7.14: Gráfico com as médias de dano para cada onda no teste com a Nave Movendo, Disparo Vermelho para as versões v1 e v3.

Semelhante a Seção 7.4.1, as Figuras 7.15, 7.16, 7.17 e 7.18 a seguir mostram o comportamento, em cada teste, do dano total de cada i -ésima onda, cada uma sendo representada num *boxplot*.

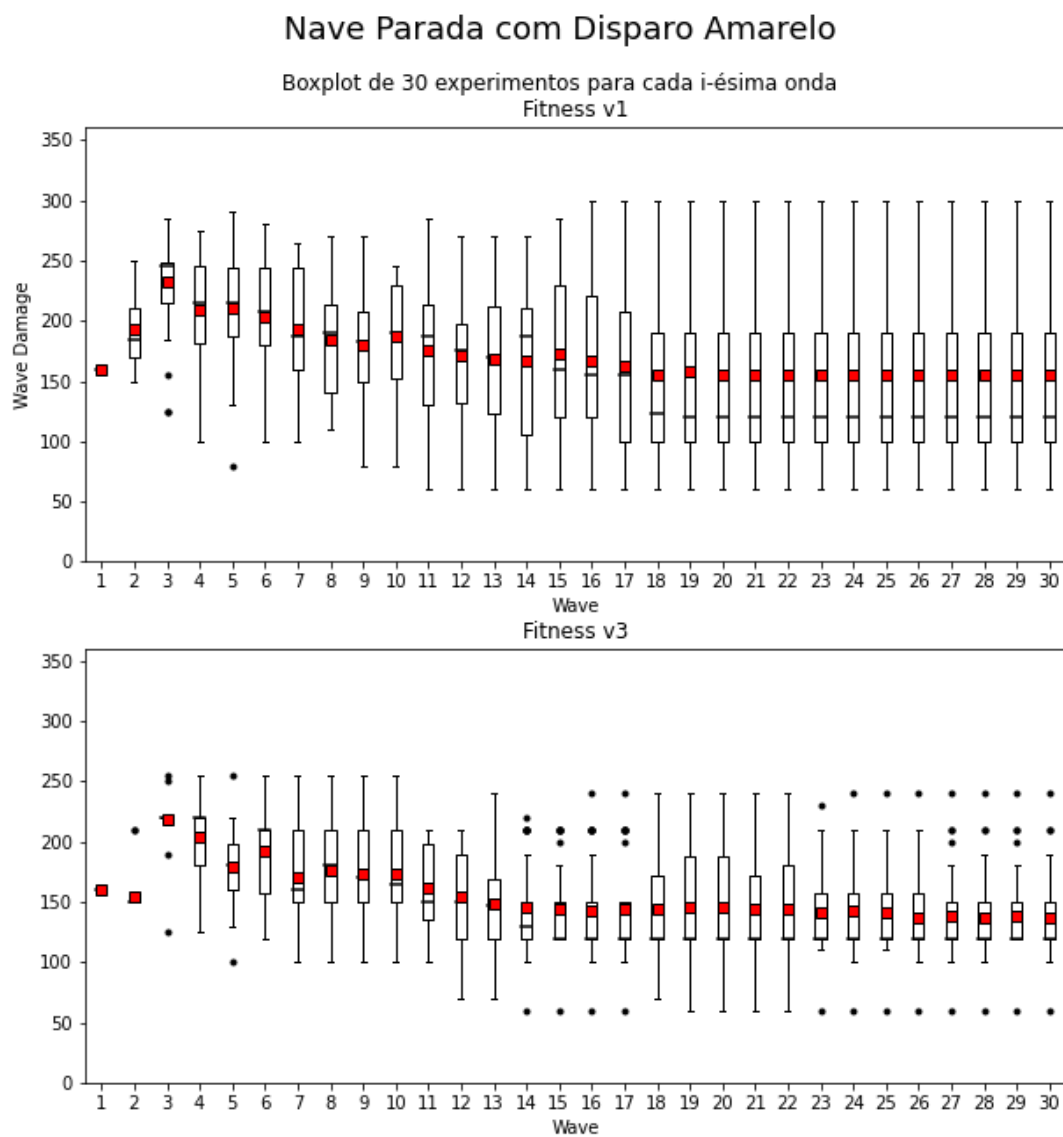


Figura 7.15: Boxplot do dano das 30 ondas nos 30 experimentos, para as versões v1 e v3.

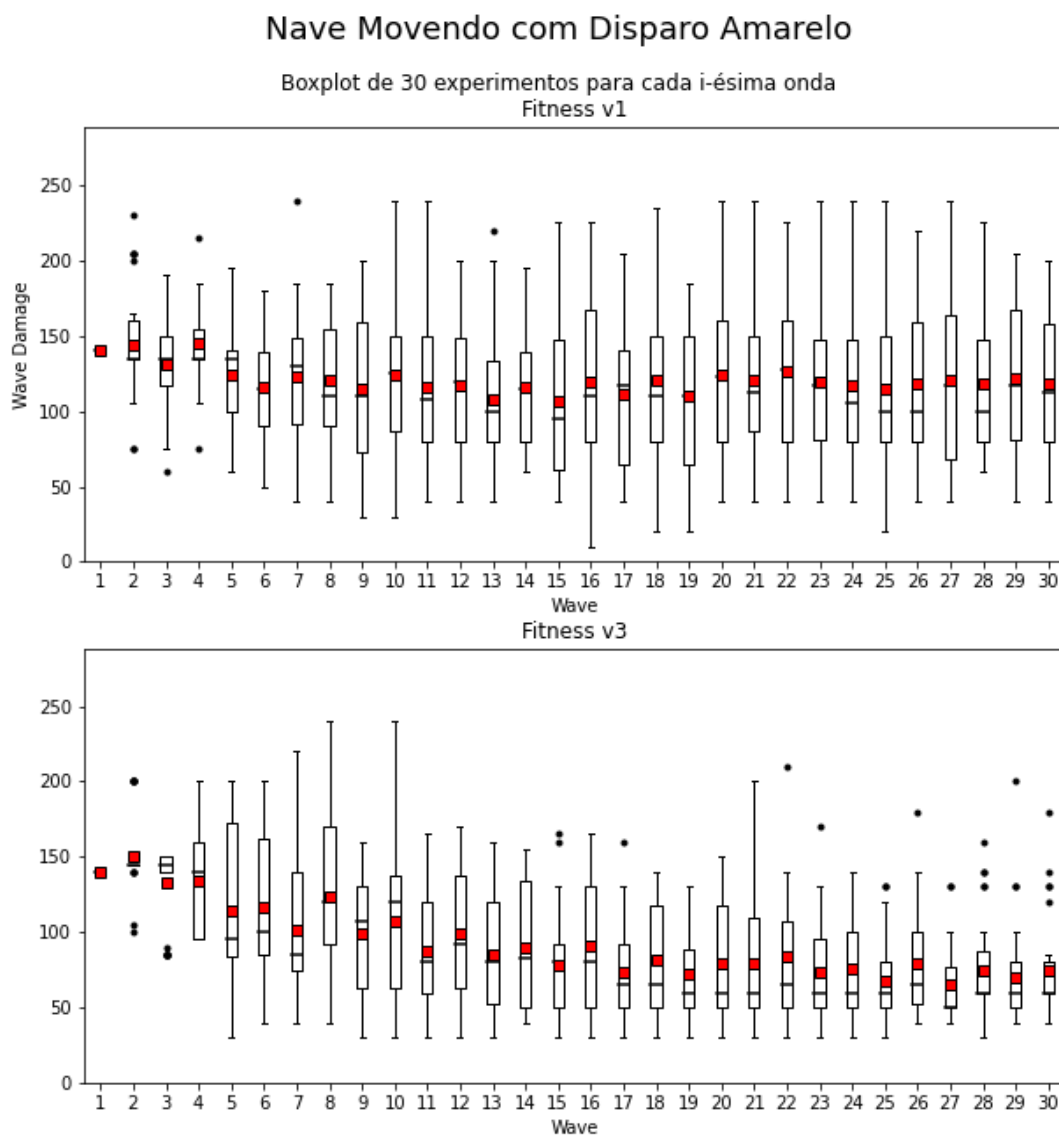


Figura 7.16: Boxplot do dano das 30 ondas nos 30 experimentos, para as versões v1 e v3.

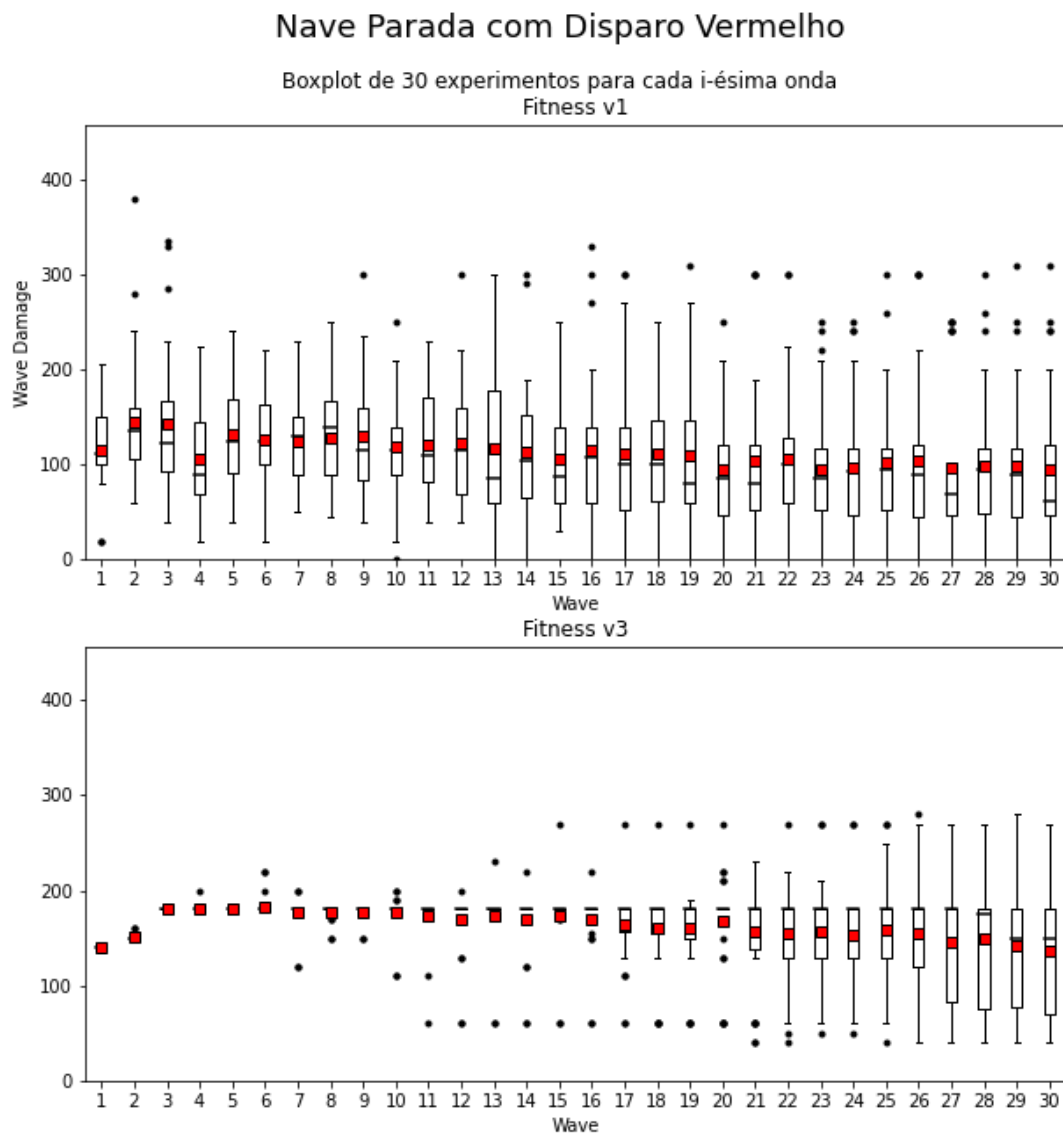


Figura 7.17: Boxplot do dano das 30 ondas nos 30 experimentos, para as versões v1 e v3.

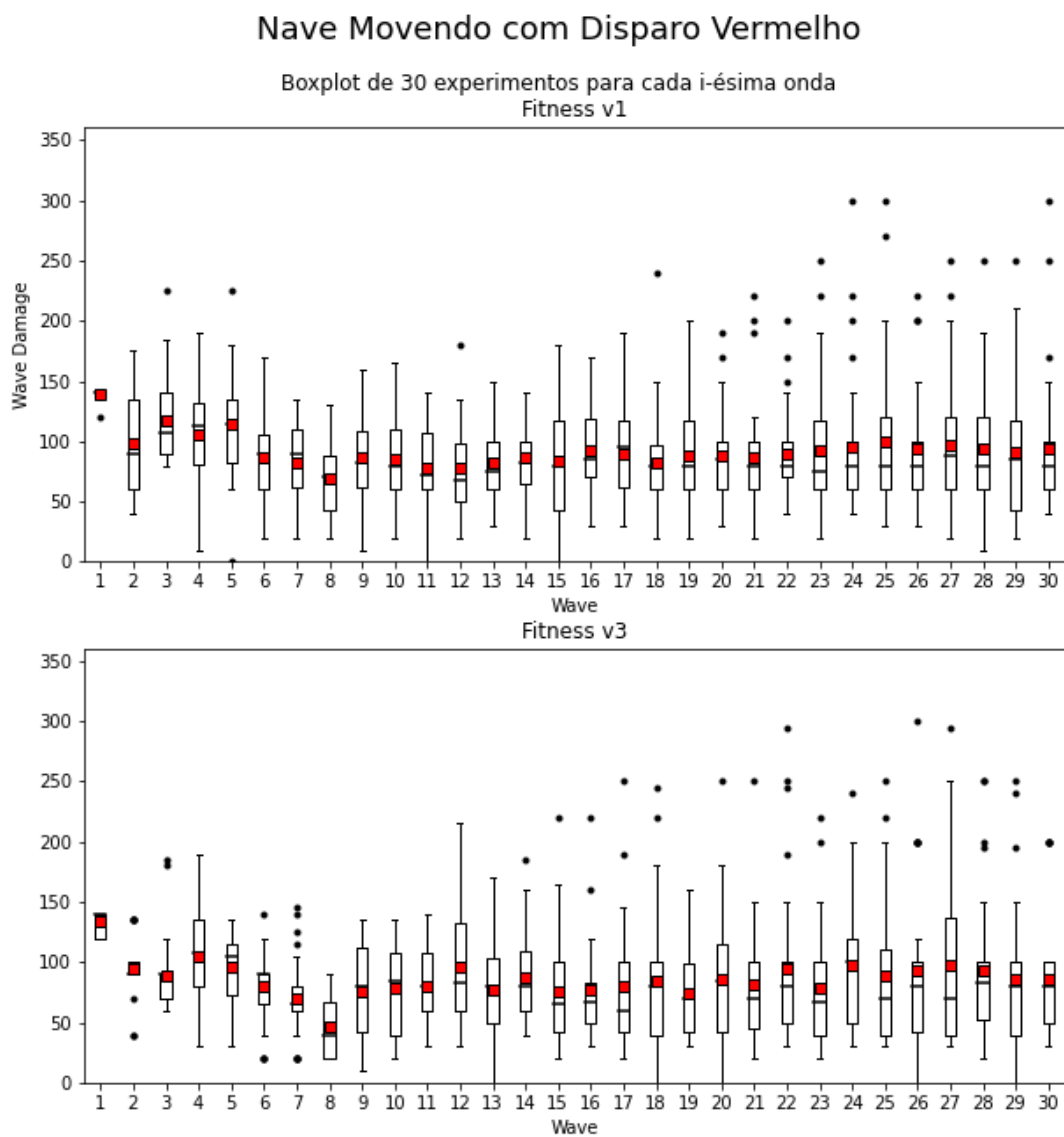


Figura 7.18: Boxplot do dano das 30 ondas nos 30 experimentos, para as versões v1 e v3.

A Tabela 7.18 mostra a onda aproximada onde o algoritmo parece ter convergido, e o dano médio após a estabilidade:

Nave	v1		v3	
	Convergiu	Dano Médio	Convergiu	Dano Médio
Disparo Amarelo Parada	18	163.54	13	147.66
Disparo Amarelo Movendo	Não	-	Não	-
Disparo Vermelho Parada	Não	-	Não	-
Disparo Vermelho Movendo	Não	-	Não	-

Tabela 7.18: Tabela mostrando a onda onde ocorreu a convergência e o dano médio a partir desse ponto até o final.

7.4.4 Space Shooter - Comparação

No *Space Shooter* nenhuma versão do algoritmo genético se mostrou adequada pois as ondas convergiram em um ponto não maximal de dano causado, e, em somente um caso, conseguiram dano maior do que na geração aleatória, nunca atingindo o dano máximo de uma dessas ondas. Individualmente, as seguintes características foram apresentadas:

v1

- Consegue obter o mesmo dano da onda aleatória em dois testes - Nave Parada com Disparo Amarelo e com Disparo Vermelho;
- Com Disparo Amarelo apresenta alta dispersão e poucos *outliers*, com média acima da mediana;
- Em particular, com a Nave Parada o algoritmo converge a um resultado não maximal, mas consegue gerar ondas de dano total próximo do máximo teórico (360) 7.2.3 (possivelmente, por priorizar os inimigos que sobrevivem mais ao invés de os que causam mais dano, uma vez que a função *fitness* não prioriza o dano que cada tipo de inimigo causa);
- Em geral estabiliza de maneira oscilante, com ondas melhores e piores de maneira sequencial, e médias acima da mediana;
- Estabiliza com oscilação e taxa de dano decrescente no teste contra a Nave Parada e Disparo Vermelho, no entanto com diversas ondas que foram totalmente eliminadas, e algumas com dano próximo do máximo (um caso interessante, pois existem alguns tipos de inimigos que sobrevivem melhor em conjunto com outros tipos, mas morrem quando sozinhos e que se reproduz na natureza também, como a co-extinção);

v3

- No geral apresenta dispersão menos uniforme do que v1, com maior presença de *outliers*;
- Com disparo amarelo mantém média bastante acima da mediana, mas não chega a produzir ondas próximas ao máximo como v1, apesar de reduzir ocorrências de inimigos fracos;
- Converge no teste com Nave Parada e Disparo Amarelo;
- Apresenta o único dano acima da onda aleatória entre todos os testes contra a Nave Parada e Disparo Vermelho, onde possui pouca variação até a onda 10, quando começam *outliers*, e a partir da 20 aumenta drasticamente a dispersão;
- Estabiliza de maneira oscilante, com taxa decrescente de dano nos testes com Nave Parada e Disparo Amarelo e Nave Movendo com Disparo Vermelho;
- Praticamente empata com o gerador aleatório no teste com a Nave Movendo e Disparo Vermelho, mas de maneira oscilante e viés decrescente, apresentando *outliers* próximos do dano máximo possível.

Ambos os *fitness* apresentam comportamento semelhante, onde as ondas iniciais melhoram os resultados coletivos - o dano total - mas progressivamente perdem eficiência, possivelmente devido a tentativa do algoritmo buscar os resultados melhores individuais mas sem a capacidade de detectar que a ordem de aparecimento potencializa o dano, pois a nave só pode atirar no asteroide que apareceu primeiro em seu raio de visão. Logo, é possível que um inimigo lento seguido de outros velozes maximizem o dano de uma rodada, ao "distrair" o jogador, mas que numa avaliação individual essa sequência de inimigos perca para outra onde um inimigo mais forte atingiu o jogador.

O Disparo Vermelho, que possui maior potência, pode ter produzido resultados fracos pois ondas que são totalmente eliminadas não oferecem oportunidade a detecção de candidatos para novas gerações, deixando o algoritmo sem opções de escolha.

Capítulo 8

Considerações Finais

8.1 Análise dos Testes

Considerando os danos máximos teóricos e os danos médios dos casos repetidos, apresentados nas seções [7.2.1](#) e [7.2.3](#); os danos das ondas aleatórias mostrados em [7.2.2](#) e [7.2.4](#); e os testes do algoritmo genético em [7.4.1](#) e [7.4.3](#), utilizando os dados após a convergência dos algoritmos, obtemos as tabelas [8.1](#) e [8.2](#), [8.3](#) e [8.4](#):

Torres Verdes	Torres Vermelhas
Máximo Calculado = 540.00	Máximo Calculado = 540.00
Fitness v3 = 482.57	Repetido EnemyGreen = 360.00
Fitness v2 = 478.61	Repetido EnemyBlue = 355.85
Repetido EnemyGreen = 450.00	Repetido EnemyRed = 180.00
Fitness v1 = 307.04	Repetido OneEach = 167.82
Repetido OneEach = 122.35	Fitness v3 = 167.44
Repetido EnemyPurple = 90.06	Fitness v1 = 157.78
Repetido EnemyRed = 89.90	Repetido EnemyPurple = 127.60
Aleatória = 81.42	Aleatória = 125.35
Repetido EnemyBlue = 60.68	Repetido EnemyYellow = 50.58
Repetido EnemyYellow = 40.07	Repetido EnemyOrange = 50.00
Repetido EnemyOrange = 39.93	Fitness v2 = Não Convergiu

Tabela 8.1: Dados agregados das médias de todos os testes, ordenado do maior para o menor no Tower Defense

Torres Verde + Vermelha	Torres Vermelha + Verde
Máximo Calculado = 540.00	Máximo Calculado = 540.00
Fitness v1 = 463.26	Fitness v1 = 459.30
Repetido EnemyGreen = 420.20	Repetido EnemyGreen = 449.85
Fitness v3 = 288.18	Fitness v3 = 399.16
Repetido EnemyBlue = 220.18	Repetido EnemyBlue = 329.82
Repetido EnemyRed = 148.50	Repetido EnemyRed = 149.90
Repetido OneEach = 148.45	Repetido OneEach = 135.05
Repetido EnemyPurple = 120.00	Repetido EnemyPurple = 120.00
Aleatória = 100.37	Aleatória = 102.37
Repetido EnemyYellow = 50.00	Repetido EnemyYellow = 50.00
Repetido EnemyOrange = 44.50	Repetido EnemyOrange = 49.45
Fitness v2 = Não Convergiu	Fitness v2 = Não Convergiu

Tabela 8.2: Dados agregados das médias de todos os testes, ordenado do maior para o menor no Tower Defense

Em apenas 3 casos no jogo *Tower Defense* o algoritmo genético foi capaz de atingir o maior dano, excluindo o dano teórico que só acontece quando o jogador não participa da partida - não faz nenhum tipo de *input* exceto iniciar o jogo. Nos testes com as torres heterogêneas (Verde + Vermelha e Vermelha + Verde) o algoritmo genético com a versão v1 ultrapassou o dano de casos repetidos e para o caso com Torres Verdes a versão v3 consegue causar dano máximo. O projeto não teve como objetivo criar um oponente imbatível, para isso seria mais simples analisar como o jogador dispôs elementos de jogo e a partir disso desenvolver métodos pré-definidos de contra-ataque; ou até mesmo sempre utilizar ondas repetidas que maximizassem o dano, mas tais estratégias poderiam tornar o jogo frustrante ou repetitivo. Fazer com que a composição das ondas de inimigos de adéquem às condições

de jogo é mais interessante, e o fato das ondas produzidas sempre superarem as ondas aleatórias neste caso mostram que uma potencial solução está sendo encontrada.

Nave Parada Disparo Amarelo	Nave Movendo Disparo Amarelo
Máximo Calculado = 360.00	Máximo Calculado = 360.00
Inimigo3 = 314.93	Inimigo3 = 213.20
Inimigo2 = 299.78	Inimigo2 = 184.61
OneEach = 82.75	Inimigo1 = 130.10
Inimigo1 = 181.50	Aleatório = 124.95
Aleatório = 181.26	OneEach = 100.85
Fitness v1 = 163.54	Inimigo4 = 80.69
Fitness v2 = 147.66	Inimigos = 67.79
Inimigo4 = 120.00	Inimigo5 = 39.23
Inimigos = 120.00	Fitness v1 = Não convergiu
Inimigo5 = 60.00	Fitness v2 = Não convergiu

Tabela 8.3: Dados agregados das médias de todos os testes, ordenado do maior para o menor no Space Shooter

Nave Parada Disparo Vermelho	Nave Movendo Disparo Vermelho
Máximo Calculado = 360.00	Máximo Calculado = 360.00
Inimigo3 = 241.60	Inimigo3 = 169.87
Inimigo2 = 240.94	Inimigo2 = 169.67
OneEach = 125.99	OneEach = 96.99
Aleatório = 123.66	Aleatório = 91.77
Inimigo4 = 106.00	Inimigo4 = 76.29
Inimigos = 102.44	Inimigos = 65.89
Inimigo5 = 62.57	Inimigo5 = 39.26
Inimigo1 = 27.60	Inimigo1 = 24.60
Fitness v1 = Não convergiu	Fitness v1 = Não convergiu
Fitness v2 = Não convergiu	Fitness v2 = Não convergiu
Repetido EnemyOrange = 39.93	Fitness v2 = Não Convergiu

Tabela 8.4: Dados agregados das médias de todos os testes, ordenado do maior para o menor no Space Shooter

Em contraponto, no jogo *Space Shooter*, nenhuma versão consegue convergir em danos acima das ondas aleatórias, apesar de demonstrar potencial para isso contra a Nave Parada com Disparo Vermelho, mostrada no Gráfico 7.13 - a taxa de dano é decrescente neste caso, então é possível que com mais ondas o algoritmo piore em relação ao aleatório. Em particular, nos testes com a Nave Parada e Disparo Amarelo (Gráfico 7.11) e Nave Parada com Disparo Vermelho (Gráfico 7.13) a versão v3 conseguiu aumentar o dano nas primeiras ondas, o que pode indicar compatibilidade em partidas curtas ou com redução mais agressiva na possibilidade de mutação, mas seriam necessários testes para confirmar

tal possibilidade. No geral, é possível que a natureza menos determinística do estilo de jogo - onde é possível se movimentar para evadir inimigos, deixar de atacar algum oponente pra focar em outro, faça com que o algoritmo genético não seja tão eficiente quanto no *Tower Defense*, que é um jogo mais estático e de comportamento bem definido - as torres não podem ser alteradas durante a onda, e todos os inimigos devem ser atacados para uma defesa efetiva. Seria possível alterar propriedades do *Space Shooter* para diminuir a aleatoriedade da partida - por exemplo, pré-definir pontos de *spawn* dos inimigos - mas acreditou-se que tais mudanças alterariam a natureza do jogo para forçar este se adequar ao projeto. Outro ponto levantado foi em relação à adequação da função *fitness* que poderia levar em conta o dano causado pelos inimigos, já que a função prioriza muito a sobrevivência ao invés desse fator importante de métrica e os inimigos que mais sobrevivem eram justamente os que causavam menos dano.

8.2 Trabalhos Futuros

Considerando a efetividade do algoritmo genético no *Tower Defense*, seria interessante estender os testes para geração de ondas com mudanças entre elas, pois num jogo normal o jogador iria adicionar ou aprimorar torres; o que implica que o algoritmo genético teria que potencialmente reiniciar o processo de convergência para uma geração ótima diferente do anterior. Também pode ser relevante considerar o dano coletivo ao invés de somente o individual dos inimigos, dando possibilidade do algoritmo detectar que está desviando de um estado maximal. Outro ponto que foi excluído por questões de tempo foram testes reais, com jogadores humanos, para obter *feedback* sobre pontos como duração de jogo e escala de dificuldade - informações como essas permitiriam ajustes e testes posteriores em áreas como mutação e outros modelos de *fitness* que seriam mais apropriados caso fossem necessários diferentes velocidade de convergência e escalada de dificuldade.

8.3 Conclusão

Com os resultados obtidos, fica aparente que o algoritmo evolutivo implementado se mostrou mais adequado ao ambiente determinístico do jogo *Tower Defense*, ao invés da aleatoriedade do *Space Shooter*, causados pelas diferentes possibilidades de tiro (depende da primeira detecção do inimigo) e em dois testes da movimentação da nave. Seria possível tornar o *Space Shooter* mais estático, com locais de *spawn* mais distribuídos e pré-definidos, mas ao mesmo tempo o *Tower Defense* deveria se tornar mais dinâmico, com as torres sendo atualizadas e adicionadas, alterando o ambiente onde o algoritmo genético está buscando evoluir. Dentro destes cenários, o projeto termina sem que seja possível verificar quais alterações seriam mais relevantes, tampouco as diversas adaptações que poderiam ser feita no código, buscando aceleração da convergência e evitando fuga de danos maximais. Ficou claro a infinidade de possibilidades a serem exploradas em um projeto com este tema.

De maneira mais pessoal, a primeira dificuldade da realização do trabalho foi a definição da proposta, posto que como grupo, cada um tinha diferentes ideias, porém eram vagas e carecendo de rigor e objetivo necessário. Para buscar desenvolver novas habilidades, além

de exercer outras obtidas durante a graduação, o desenvolvimento de uma inteligência artificial para jogos, assim como o desenvolvimento dos jogos em si, pareceu ser uma proposta interessante. Inicialmente o projeto pareceu simples, mas com o decorrer do tempo o entendimento do problema não se mostrou trivial, e foi necessário algum tempo para compreensão da profundidade da proposta. Tempo excessivo foi gasto na obtenção de fontes e literatura sobre o tema, mas talvez parte dele pudesse ter sido melhor aproveitado na implementação, que se alongou para familiarização de todos os membros com o *Godot* e depois para o desenvolvimento dos jogos. Com os jogos prontos e o algoritmo prototipado, a definição dos testes e coleta de dados se mostrou relativamente demorada, mas foram um dos estágios mais interessantes do trabalho, pois a existência de métricas para qualificar o algoritmo se mostrou muito útil e interessante. Apesar do desejo de aprimorar mais o algoritmo e os jogos, não há tempo hábil para o mesmo, mas a experiência se mostrou edificante e enriquecedora, e demonstrou bem como desenvolver um experimento com algum rigor científico.

Apêndice A

Moda das Ondas no Tower Defense para versão v1

Foram calculadas as modas das ondas do *fitness* desenvolvido, para permitir a visualização dos inimigos mais comuns que o algoritmo convergiu.

A.1 Torres Verdes

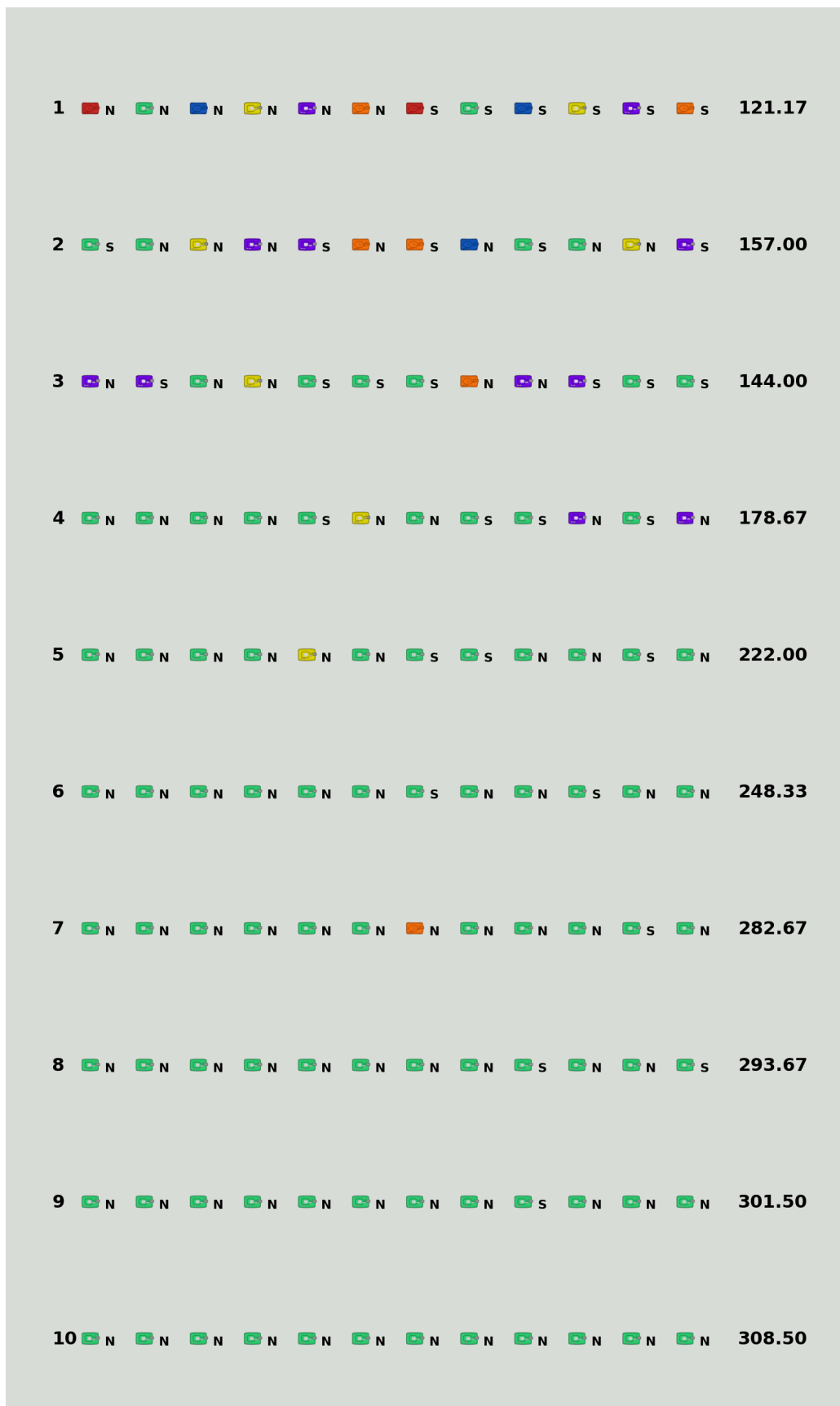


Figura A.1: Visualização da moda de cada onda com a versão v1 contra Torres Verdes.

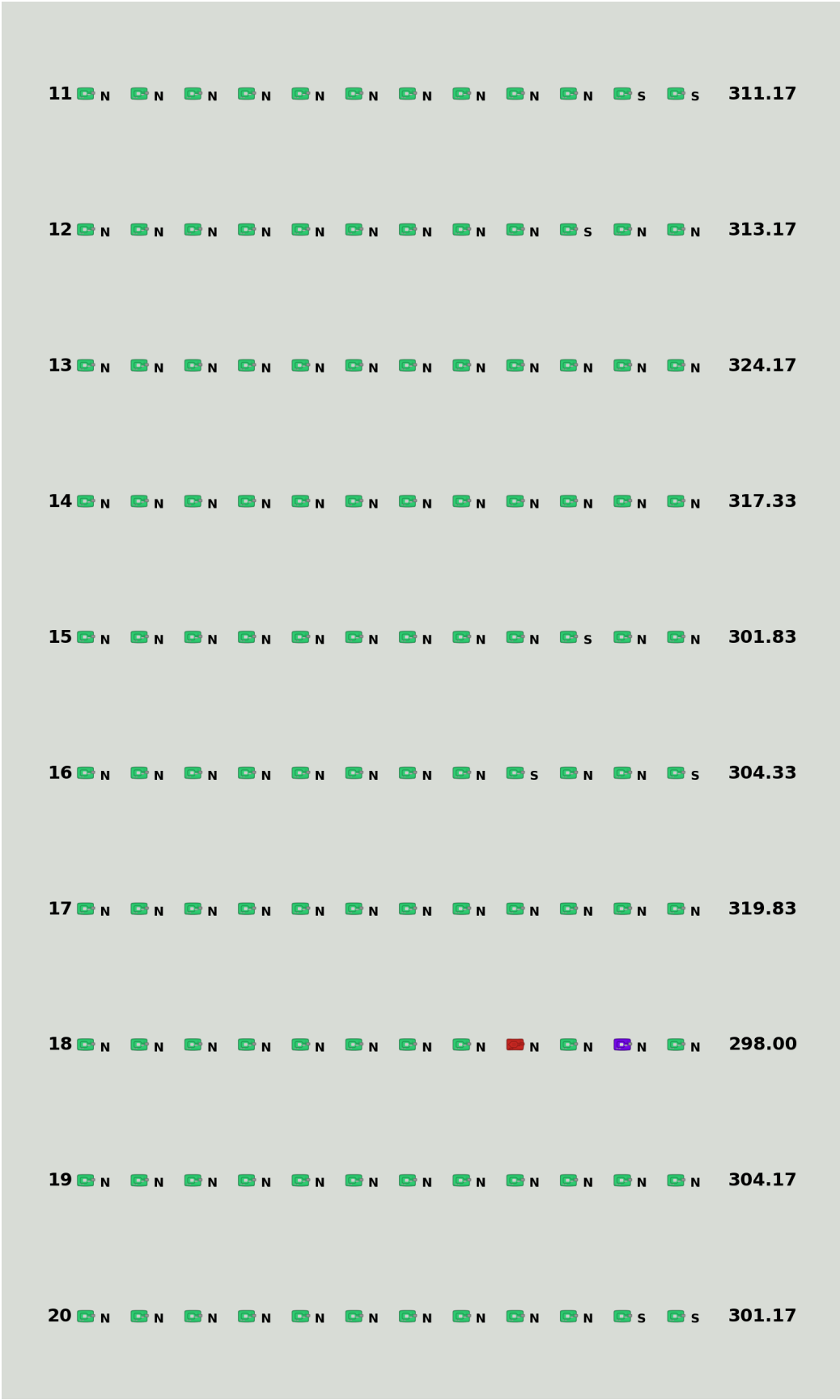


Figura A.2: Visualização da moda de cada onda com a versão v1 contra Torres Verdes.

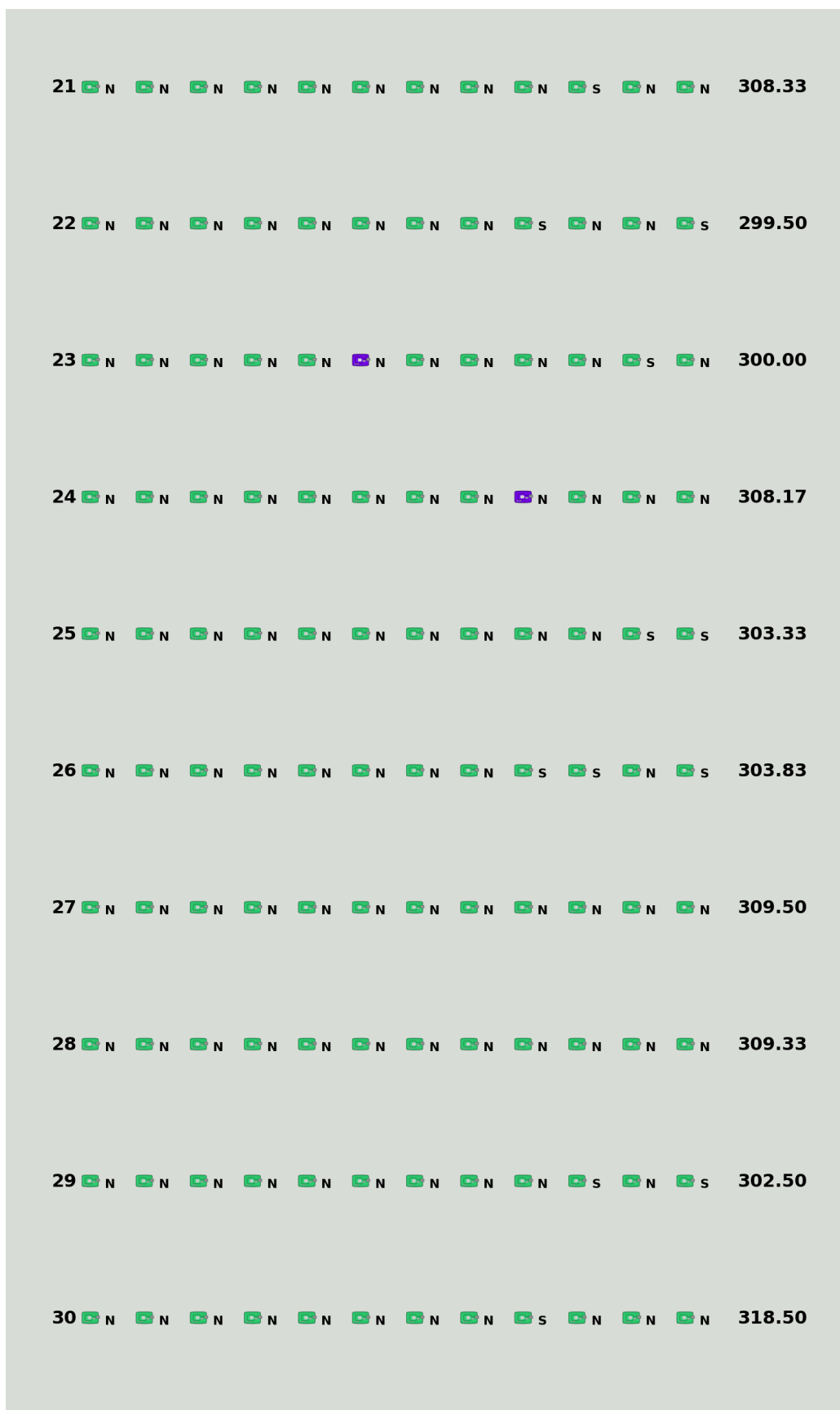


Figura A.3: Visualização da moda de cada onda com a versão v1 contra Torres Verdes.

A.2 Torres Vermelhas

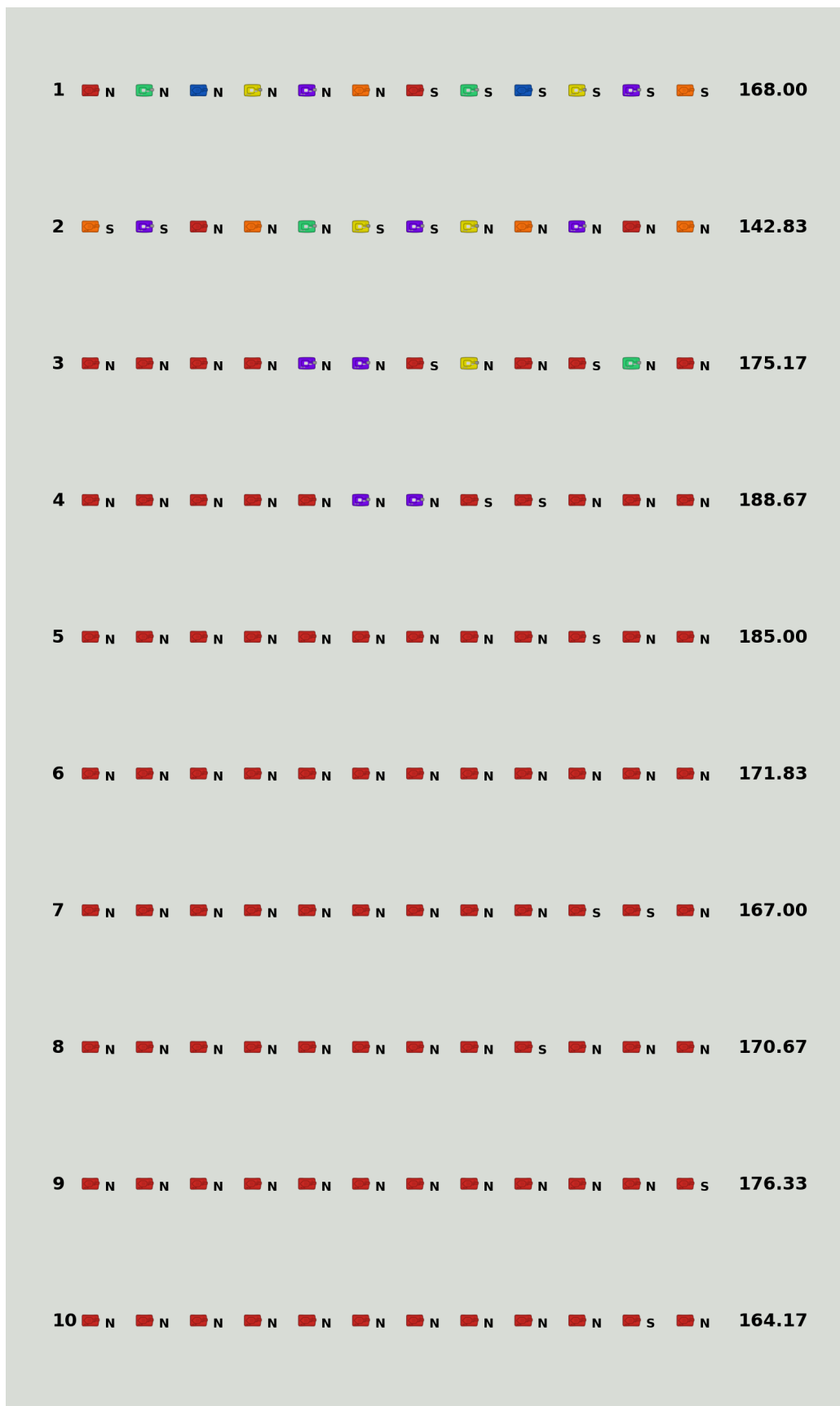


Figura A.4: Visualização da moda de cada onda com a versão v1 contra Torres Vermelhas.

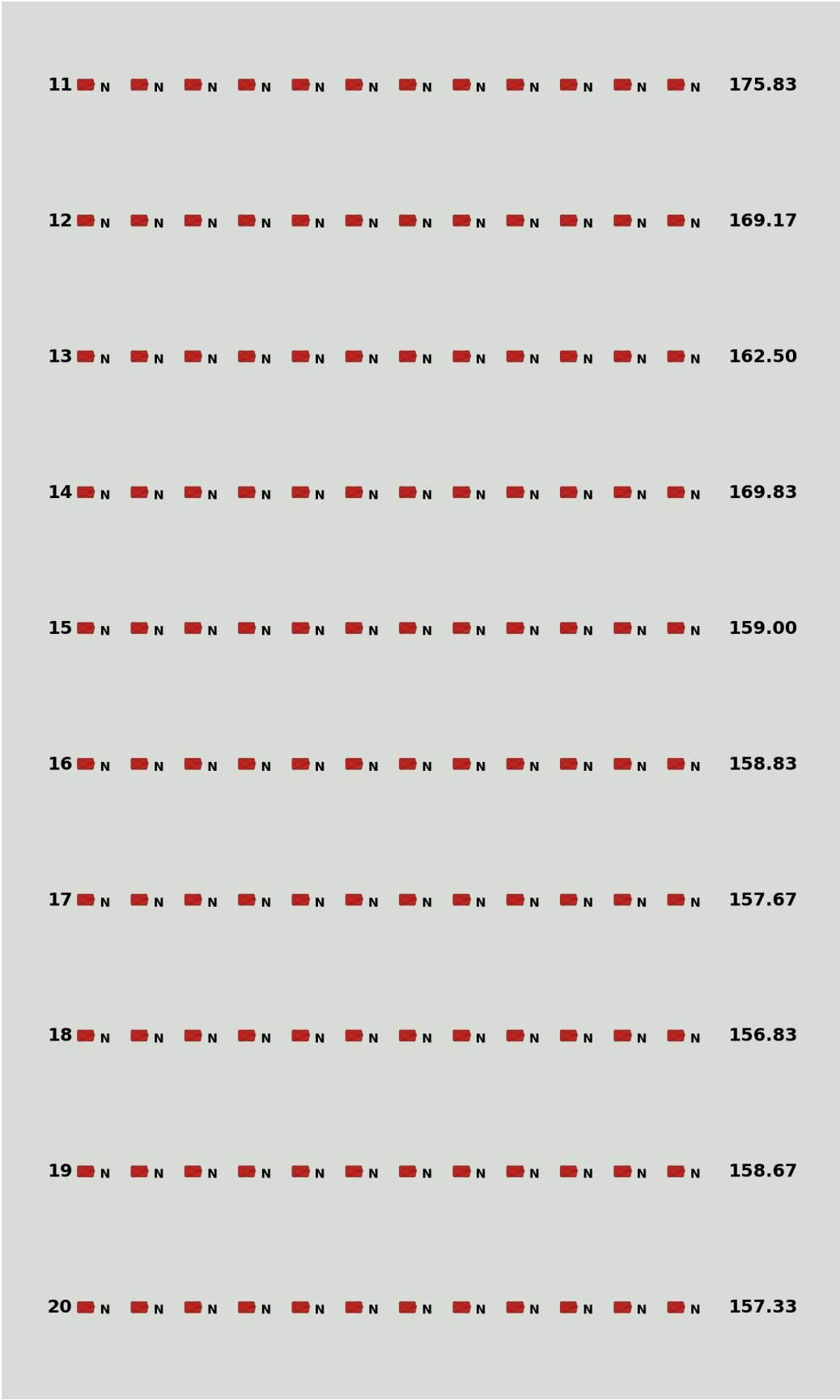


Figura A.5: Visualização da moda de cada onda com a versão v1 contra Torres Vermelhas.



Figura A.6: Visualização da moda de cada onda com a versão v1 contra Torres Vermelhas.

A.3 Torres Verde + Vermelha

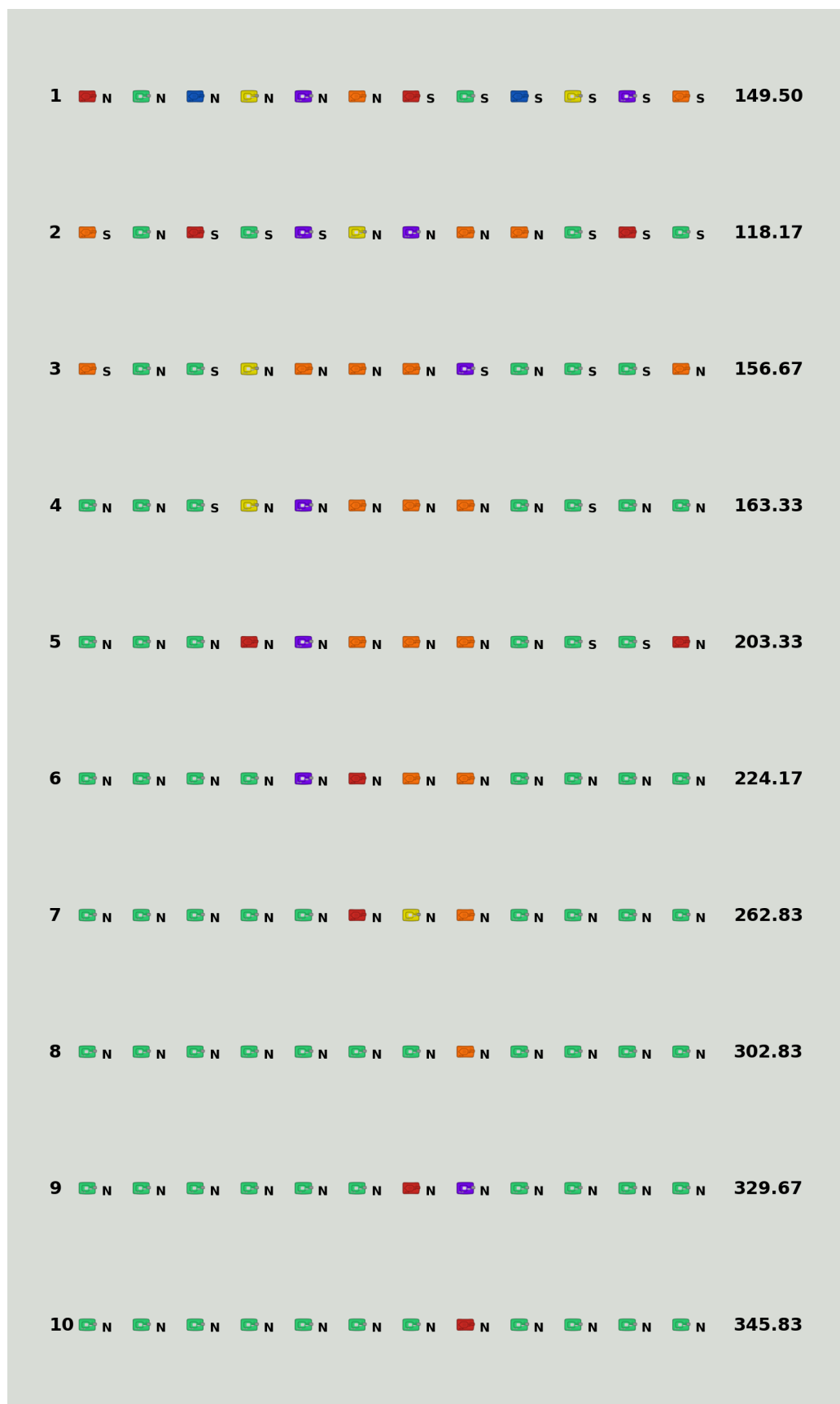


Figura A.7: Visualização da moda de cada onda com a versão v1 contra Torres Verdes + Vermelhas.



Figura A.8: Visualização da moda de cada onda com a versão v1 contra Torres Verdes + Vermelhas.

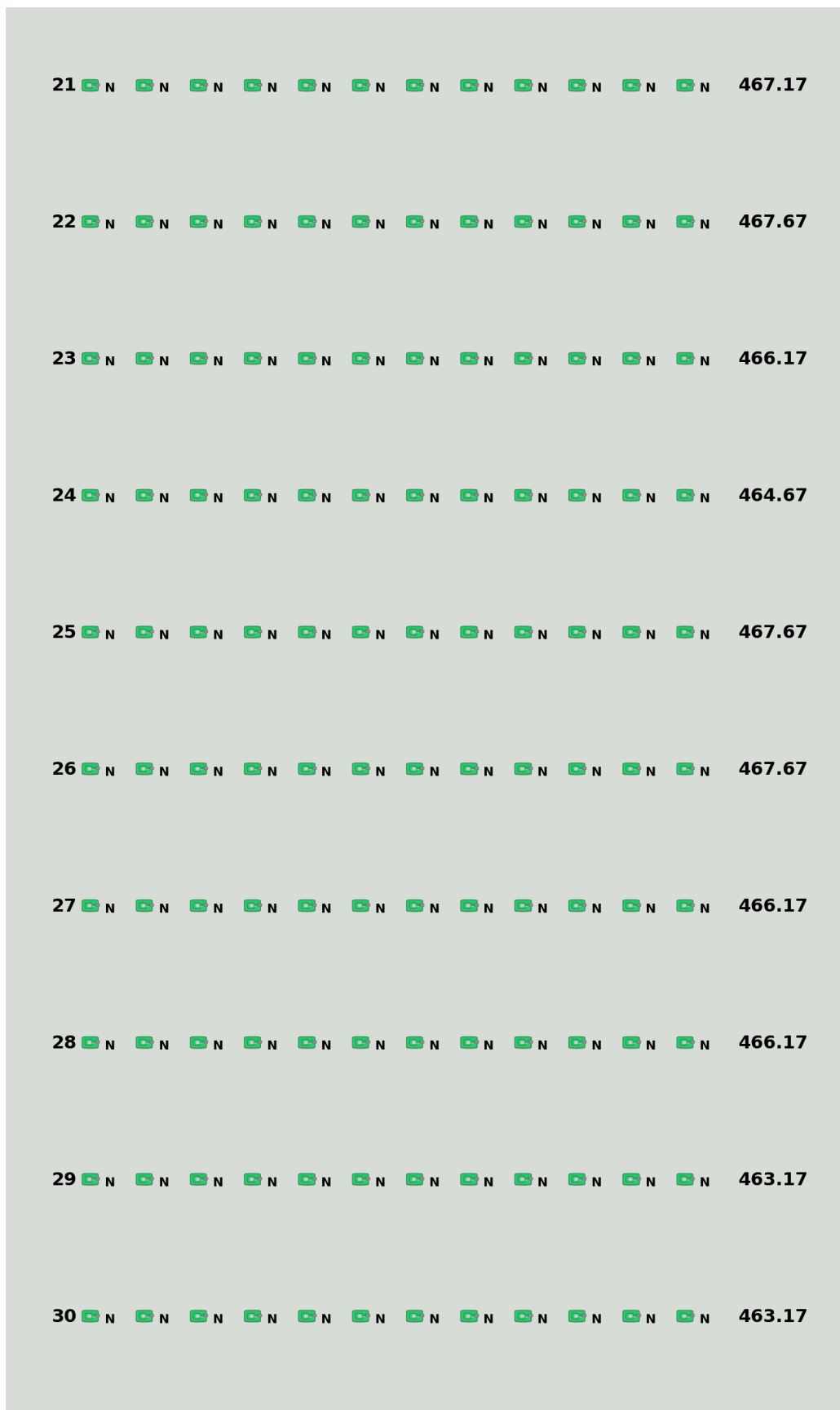


Figura A.9: Visualização da moda de cada onda com a versão v1 contra Torres Verdes + Vermelhas.

A.4 Torres Vermelha + Verde

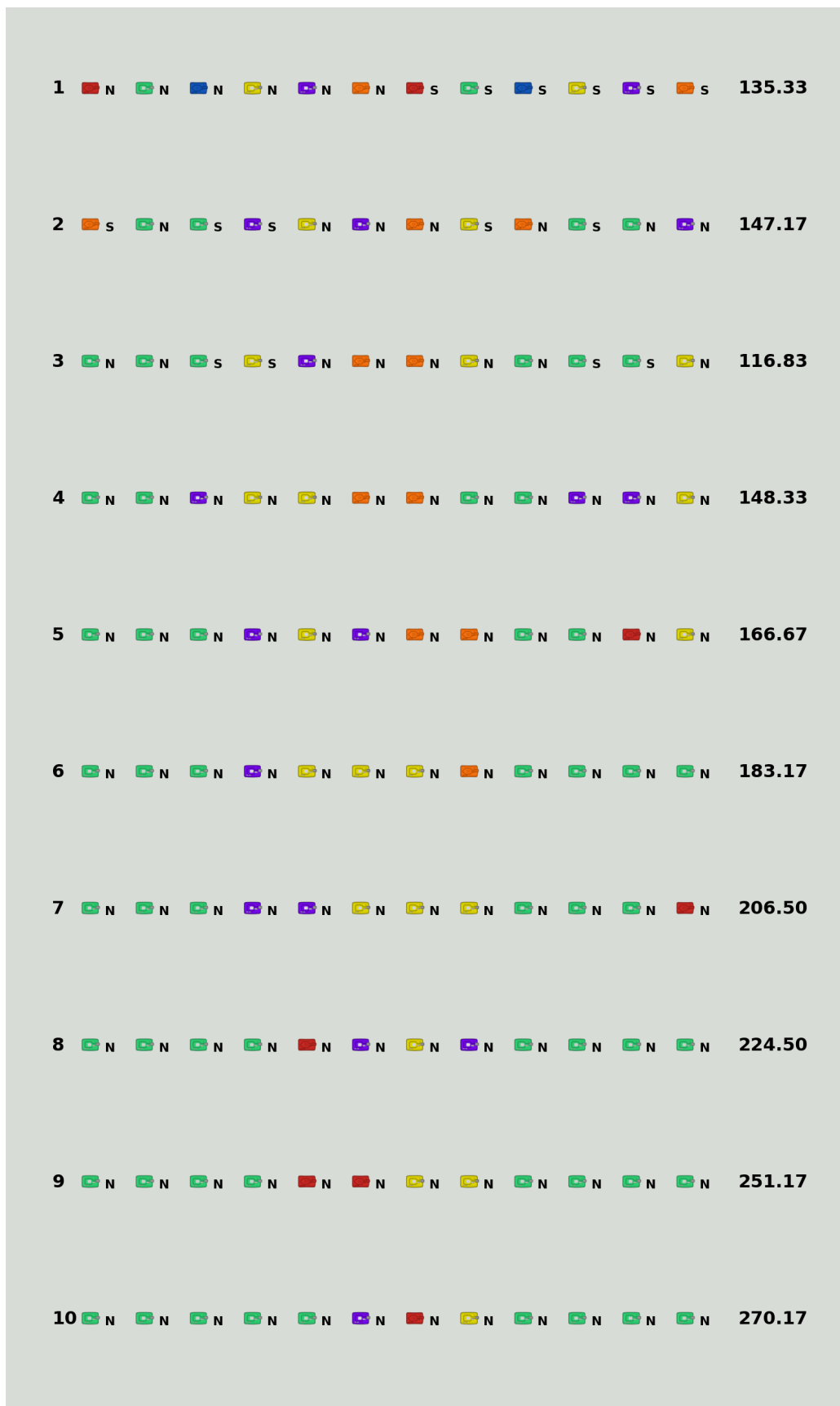


Figura A.10: Visualização da moda de cada onda com a versão v1 contra Torres Vermelhas + Verdes.

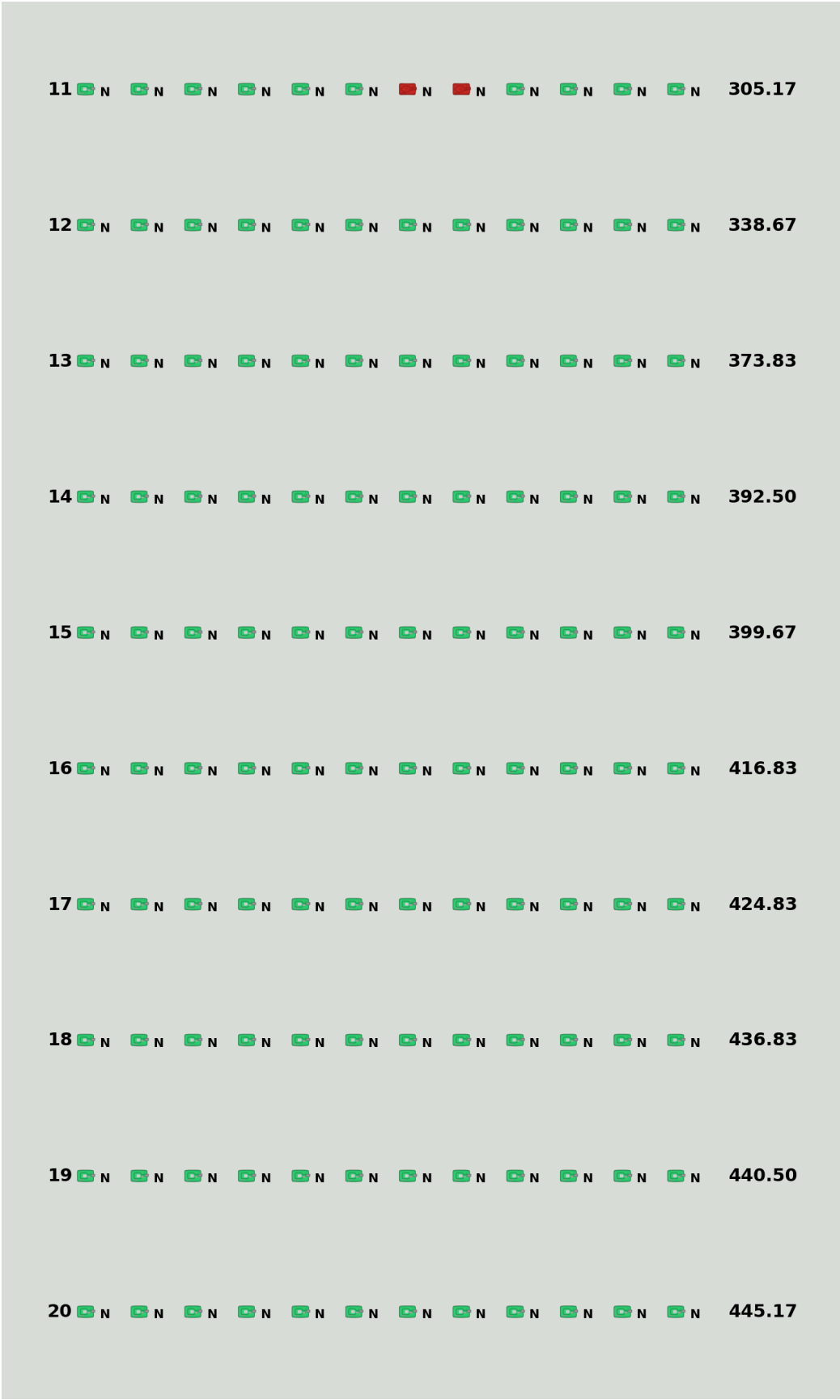


Figura A.11: Visualização da moda de cada onda com a versão v1 contra Torres Vermelhas + Verdes.

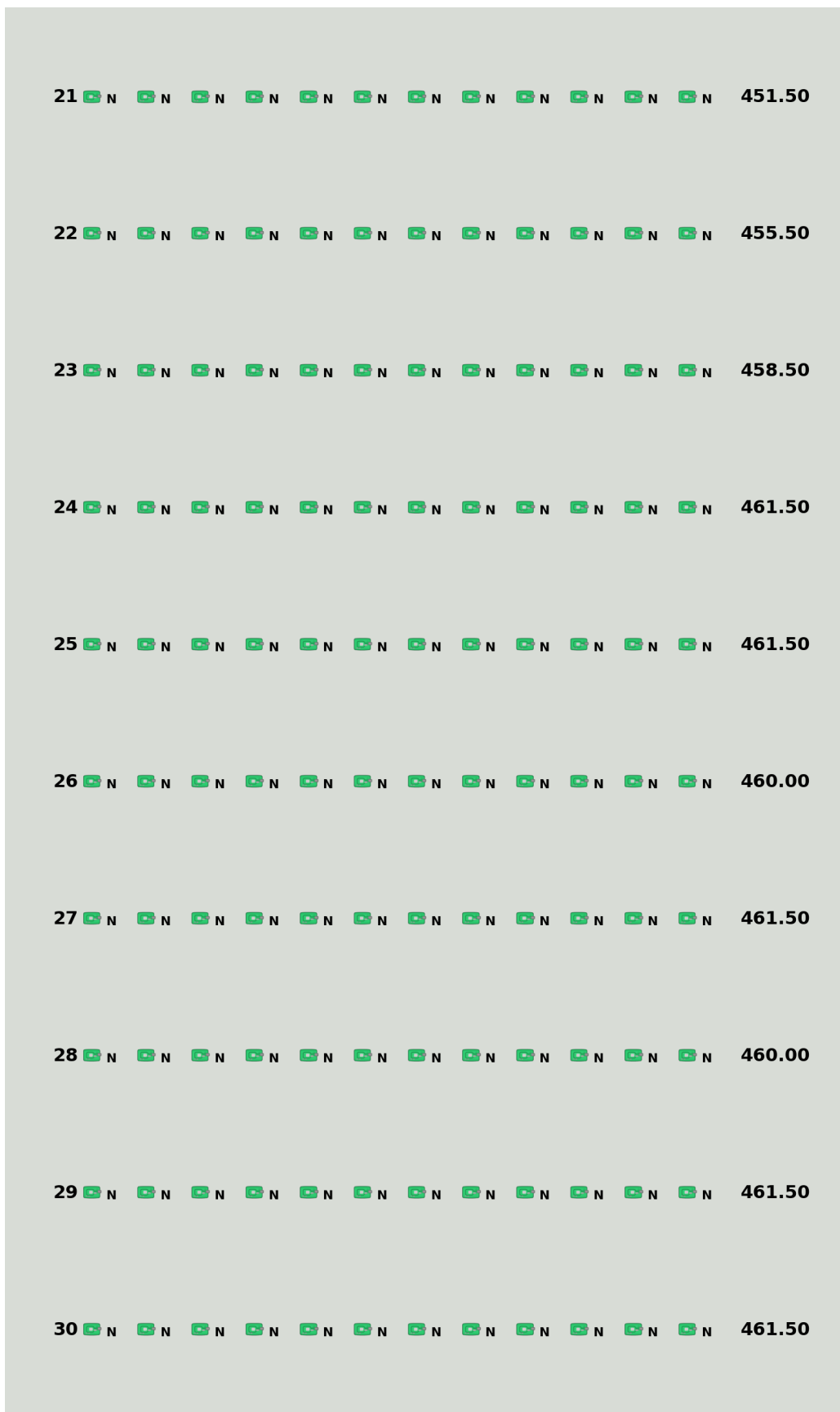


Figura A.12: Visualização da moda de cada onda com a versão v1 contra Torres Vermelhas + Verdes.

Apêndice B

Moda das Ondas no Tower Defense para o v2

Foram calculadas as modas das ondas do *fitness* desenvolvido, para permitir a visualização dos inimigos mais comuns que o algoritmo convergiu.

B.1 Torres Verdes

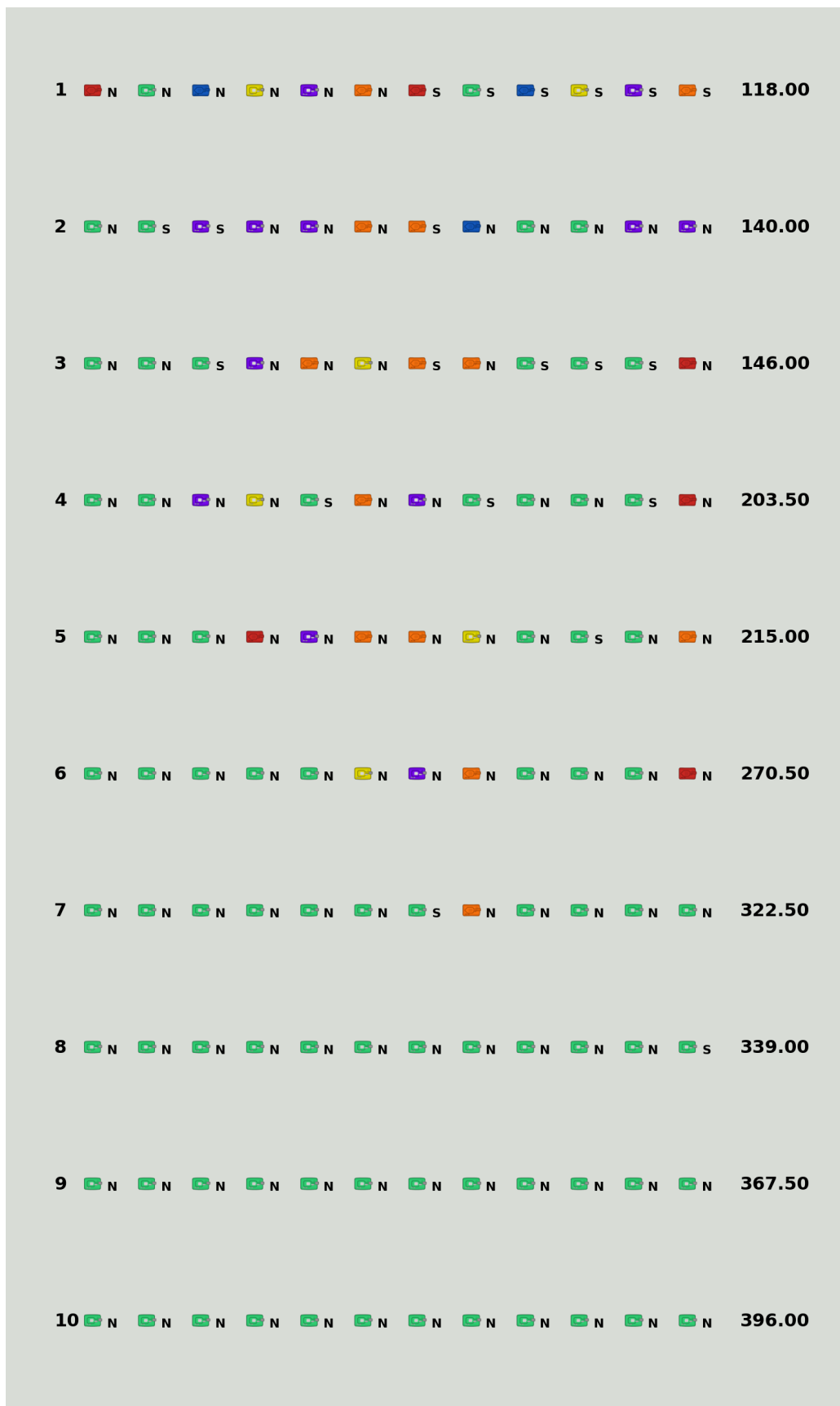


Figura B.1: Visualização da moda de cada onda com a versão v2 contra Torres Verdes.

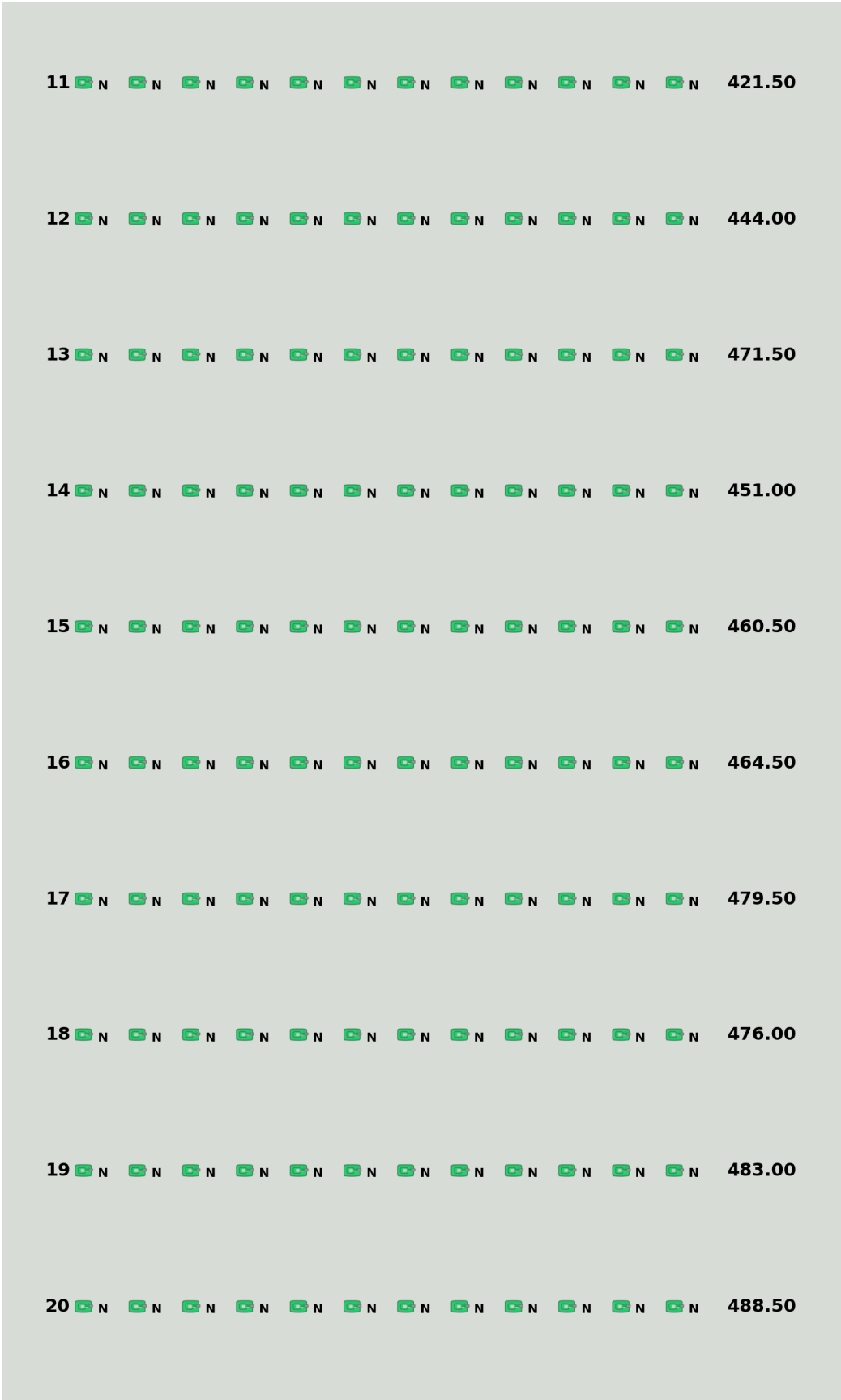


Figura B.2: Visualização da moda de cada onda com a versão v2 contra Torres Verdes.

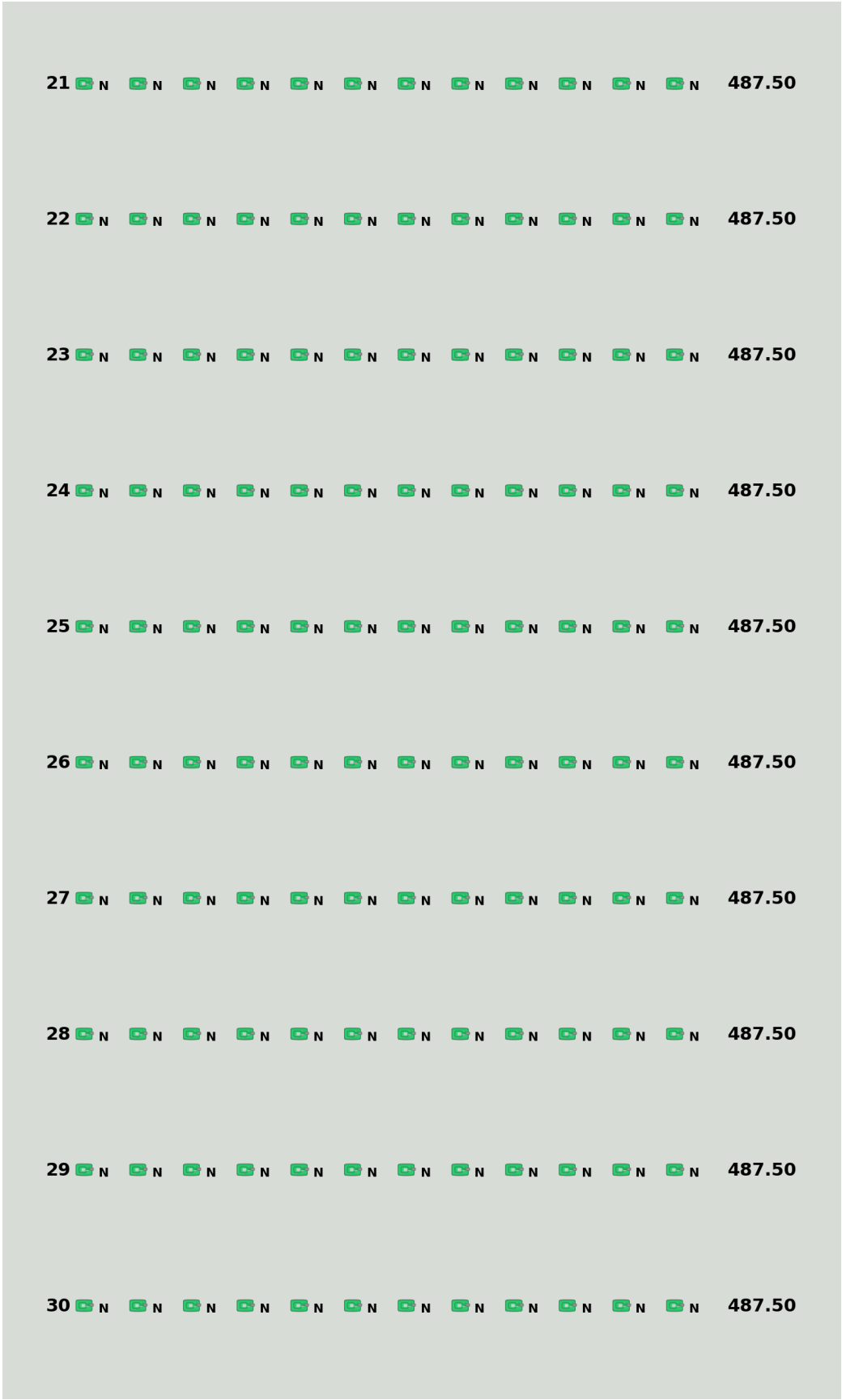


Figura B.3: Visualização da moda de cada onda com a versão v2 contra Torres Verdes.

B.2 Torres Vermelhas

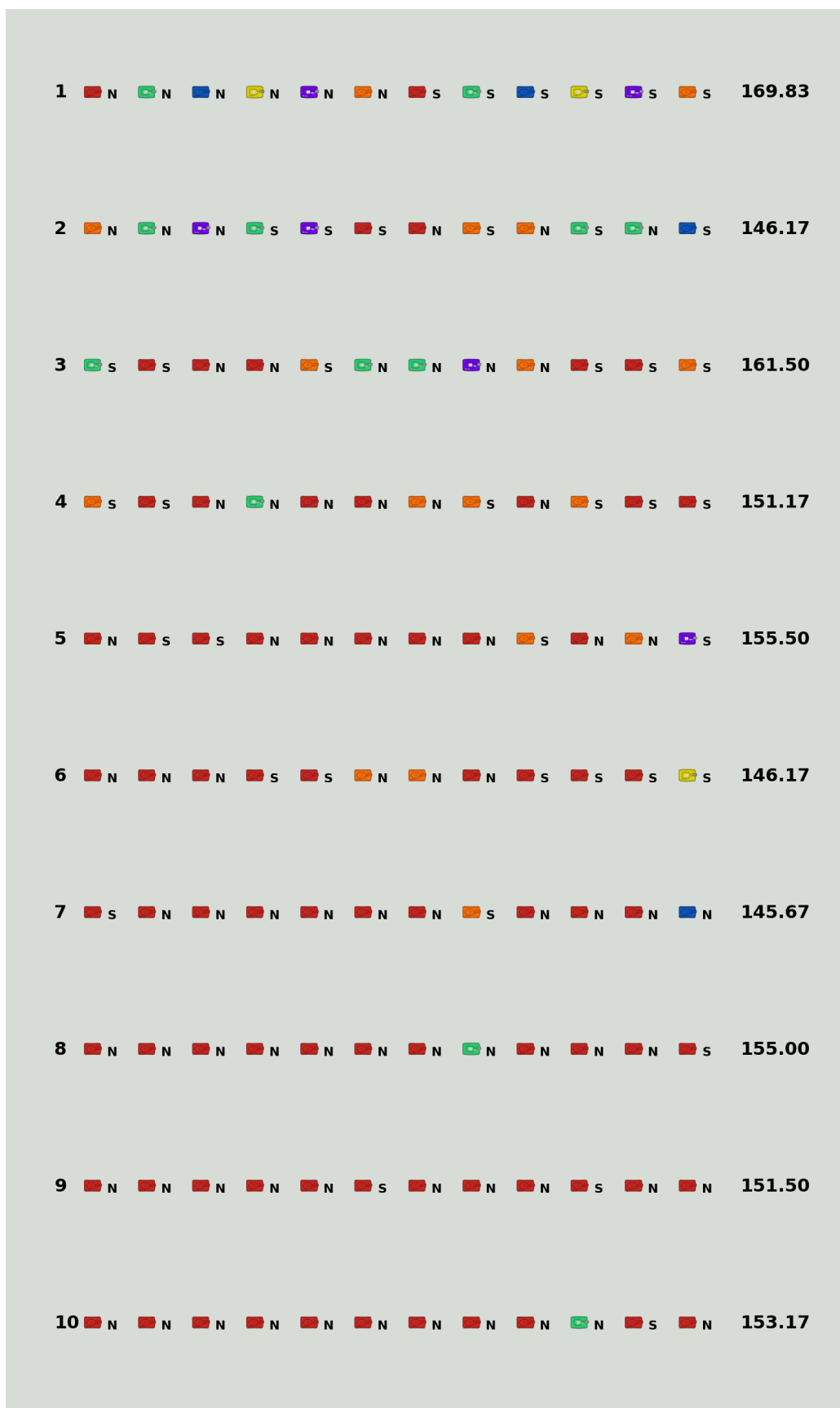


Figura B.4: Visualização da moda de cada onda com a versão v2 contra Torres Vermelhas.

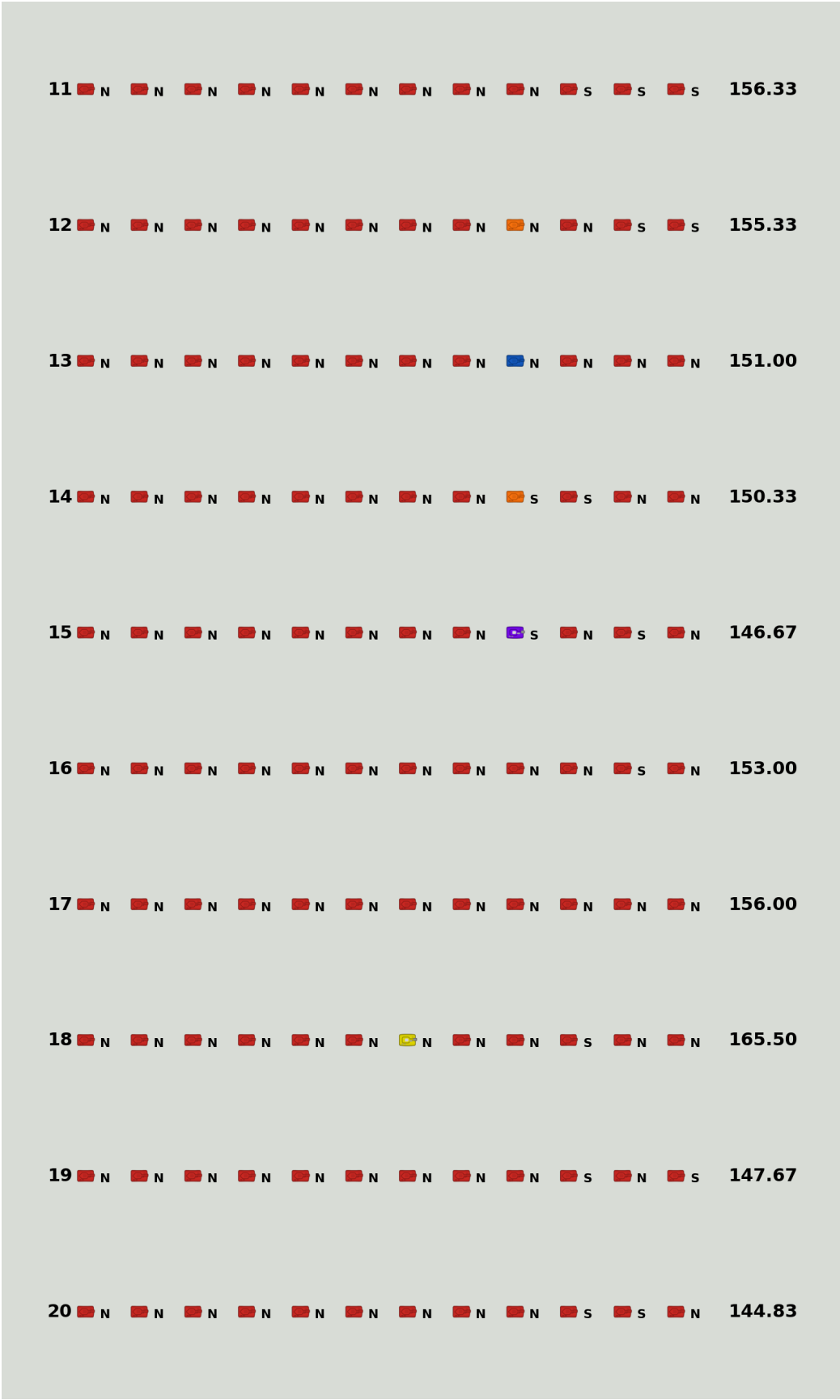


Figura B.5: Visualização da moda de cada onda com a versão v2 contra Torres Vermelhas.

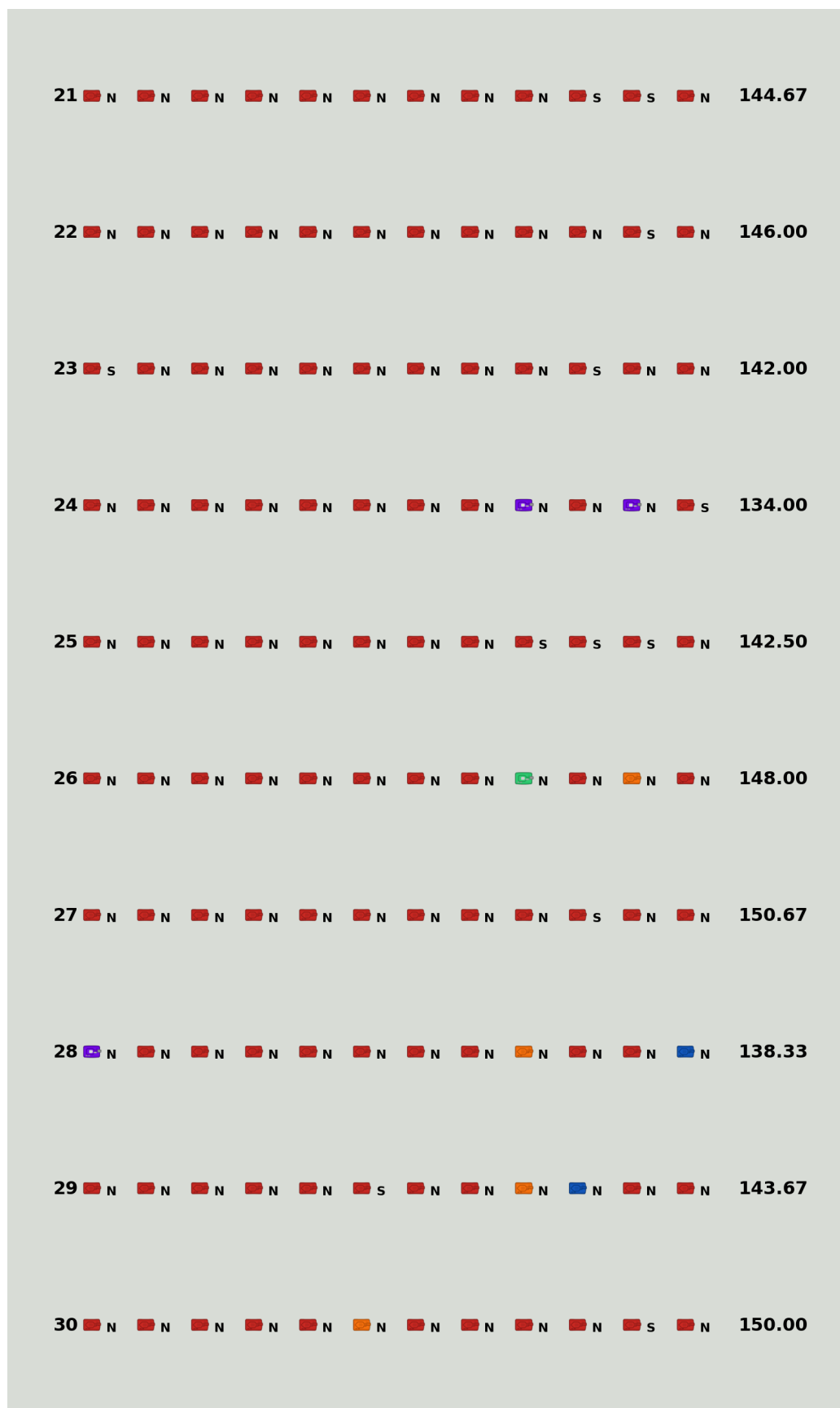


Figura B.6: Visualização da moda de cada onda com a versão v2 contra Torres Vermelhas.

B.3 Torres Verde + Vermelha



Figura B.7: Visualização da moda de cada onda com a versão v2 contra Torres Verdes + Vermelhas.

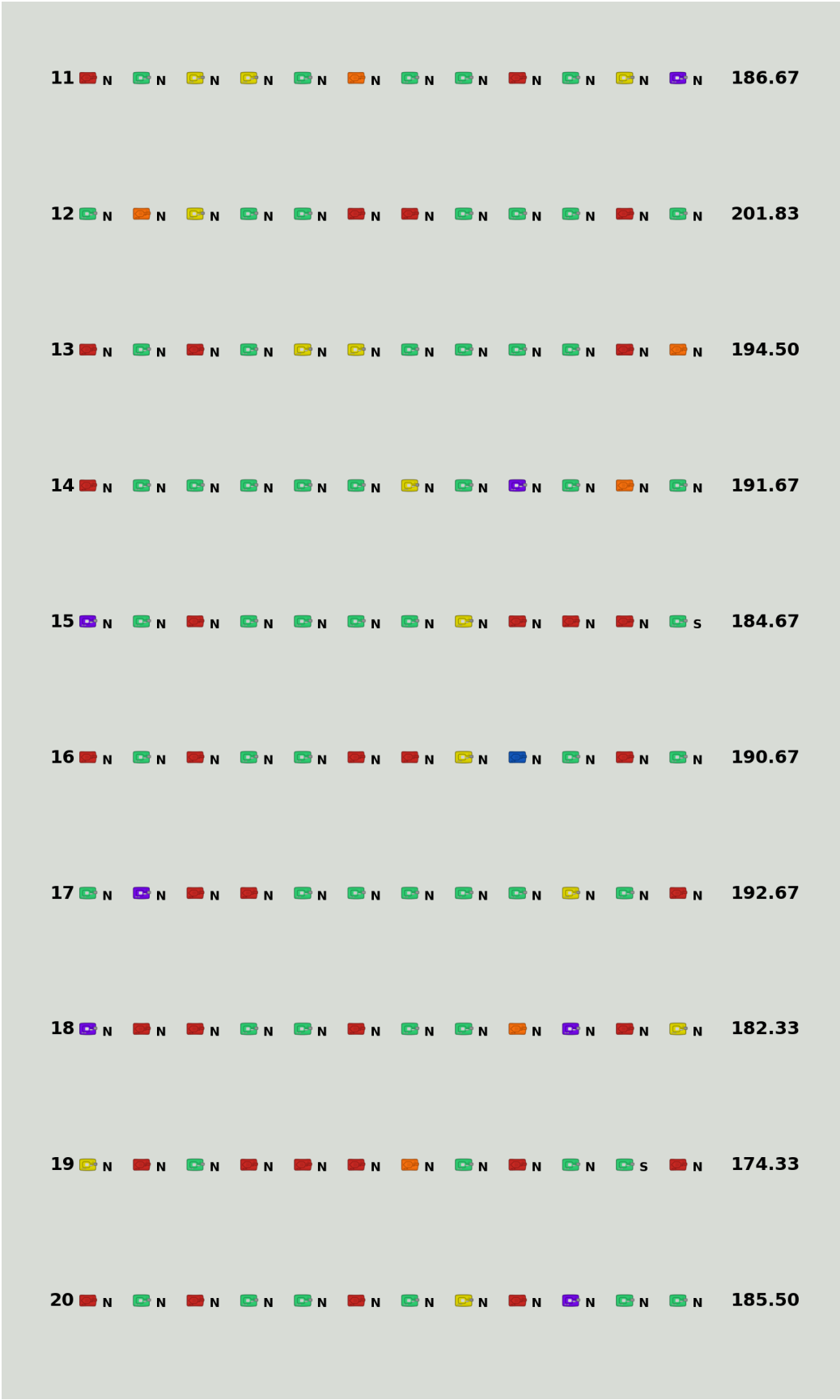


Figura B.8: Visualização da moda de cada onda com a versão v2 contra Torres Verdes + Vermelhas.

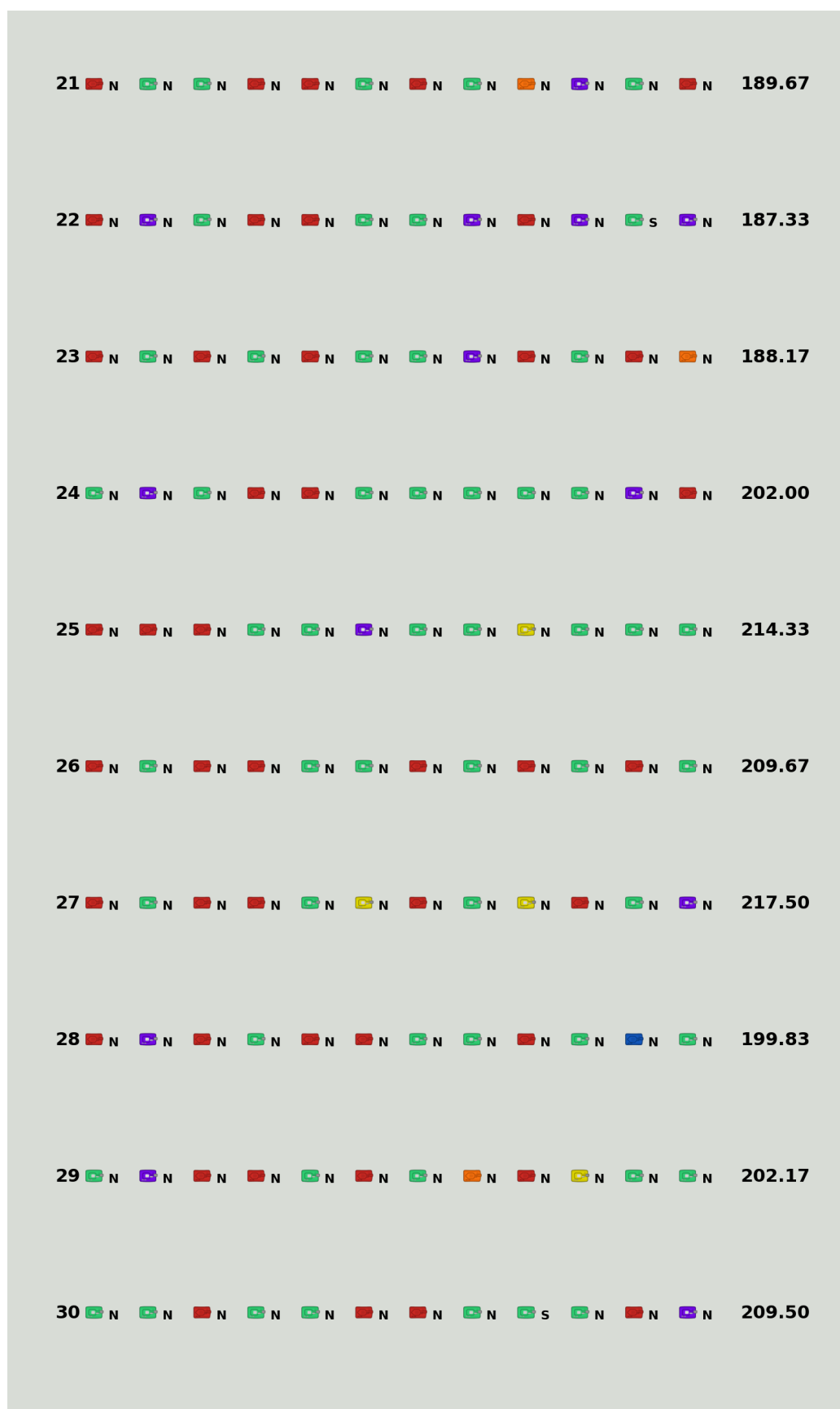


Figura B.9: Visualização da moda de cada onda com a versão v2 contra Torres Verdes + Vermelhas.

B.4 Torres Vermelha + Verde

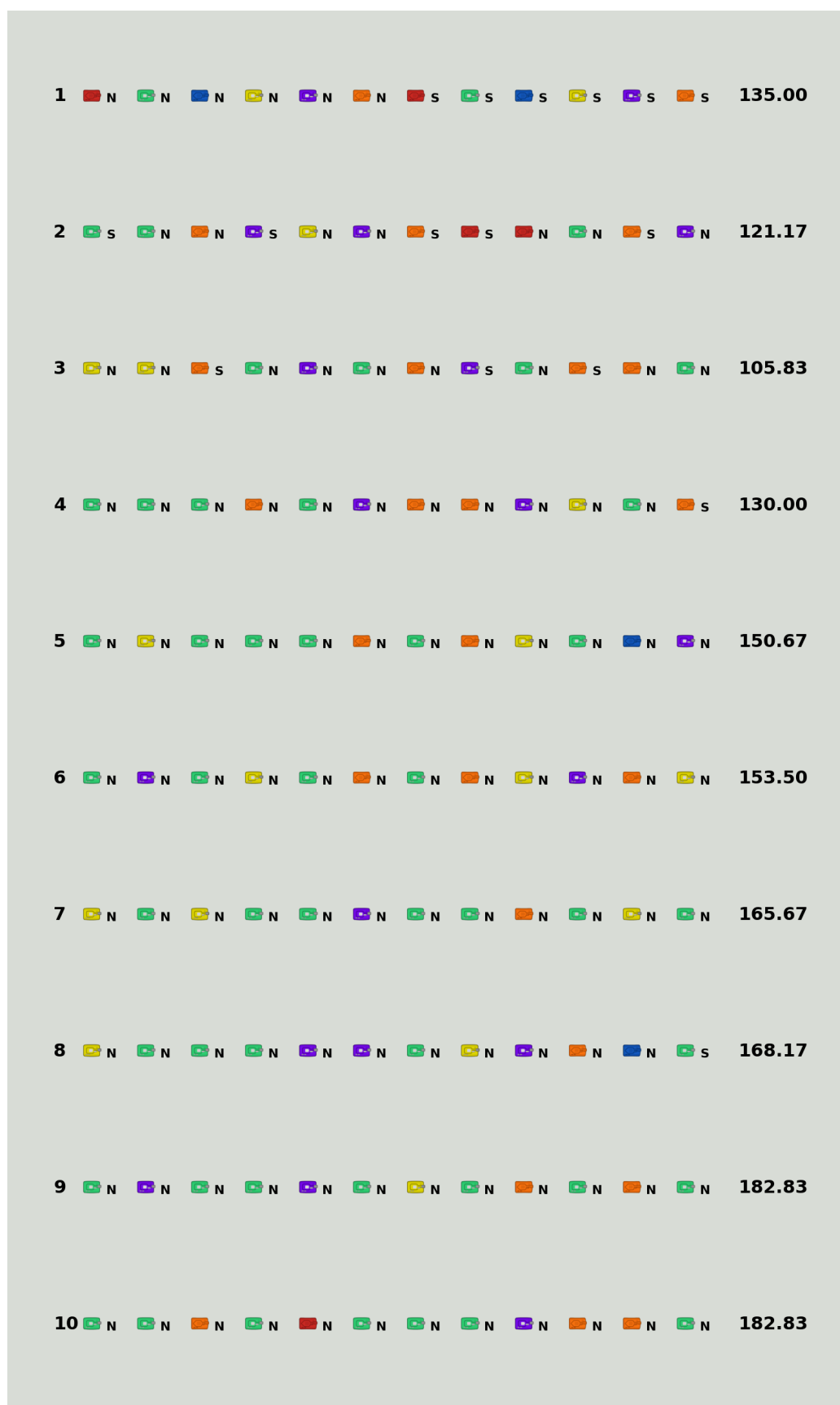


Figura B.10: Visualização da moda de cada onda com a versão v2 contra Torres Vermelhas + Verdes.

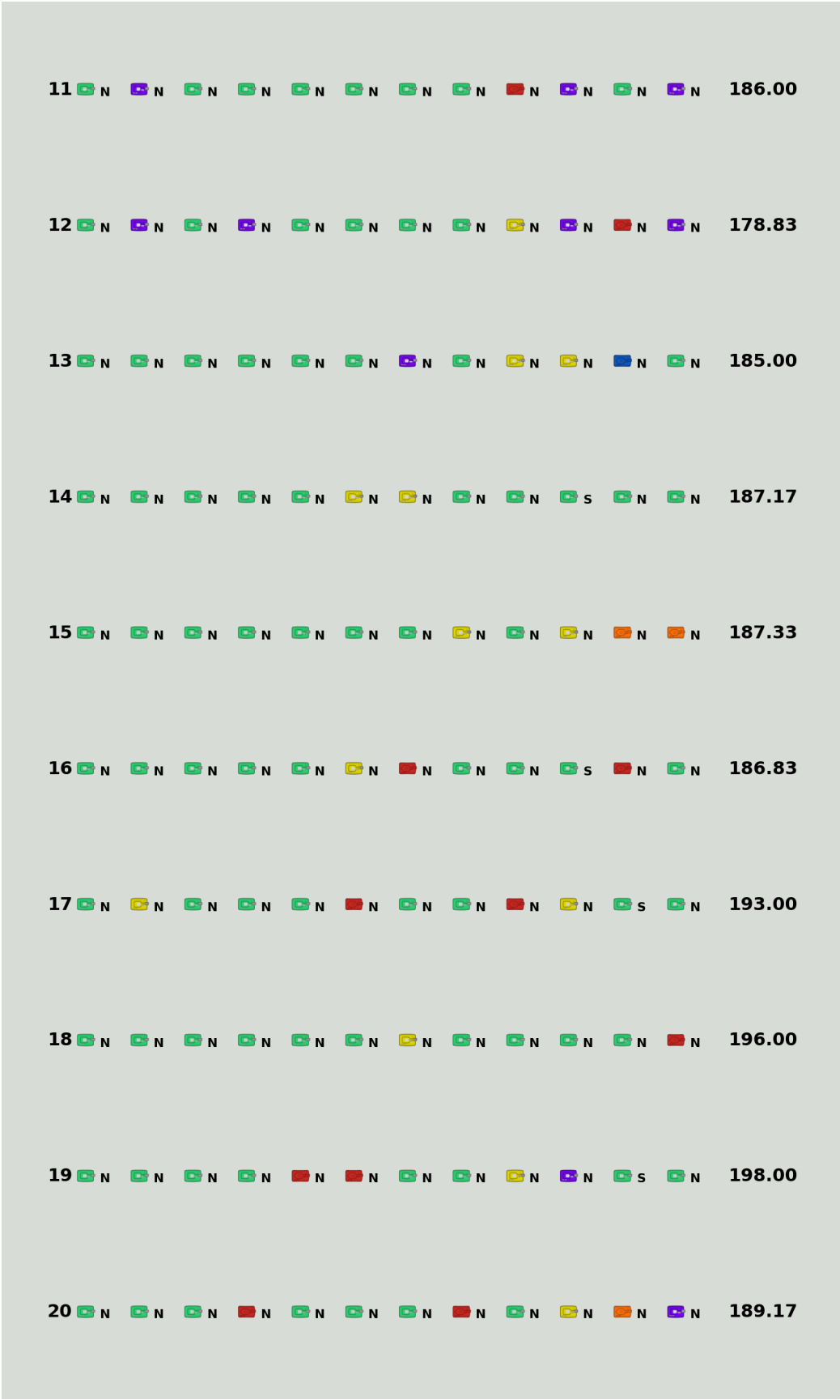


Figura B.11: Visualização da moda de cada onda com a versão v2 contra Torres Vermelhas + Verdes.

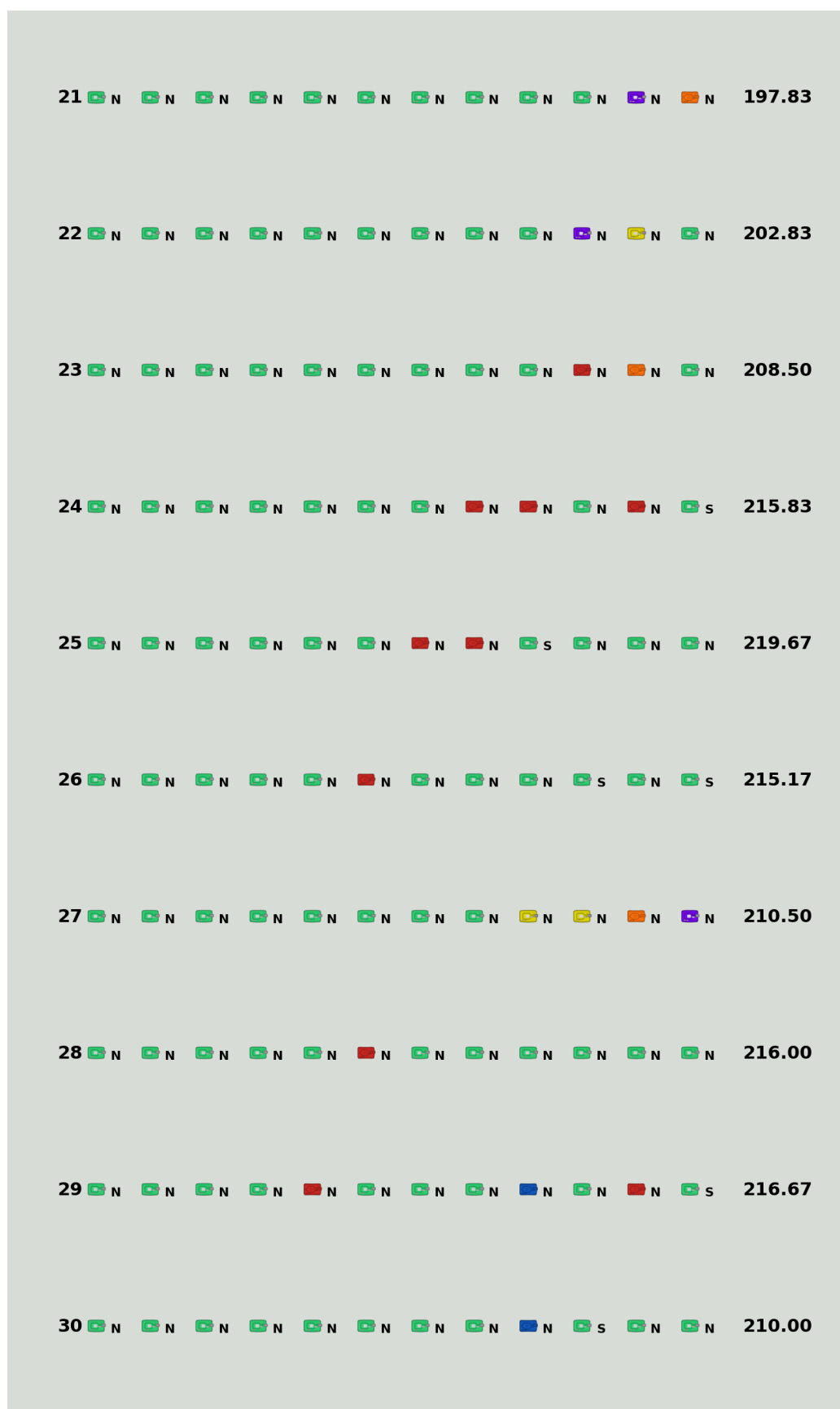


Figura B.12: Visualização da moda de cada onda com a versão v2 contra Torres Vermelhas + Verdes.

Apêndice C

Moda das Ondas no Tower Defense para a versão v3

Foram calculadas as modas das ondas do *fitness* desenvolvido, para permitir a visualização dos inimigos mais comuns que o algoritmo convergiu.

C.1 Torres Verdes

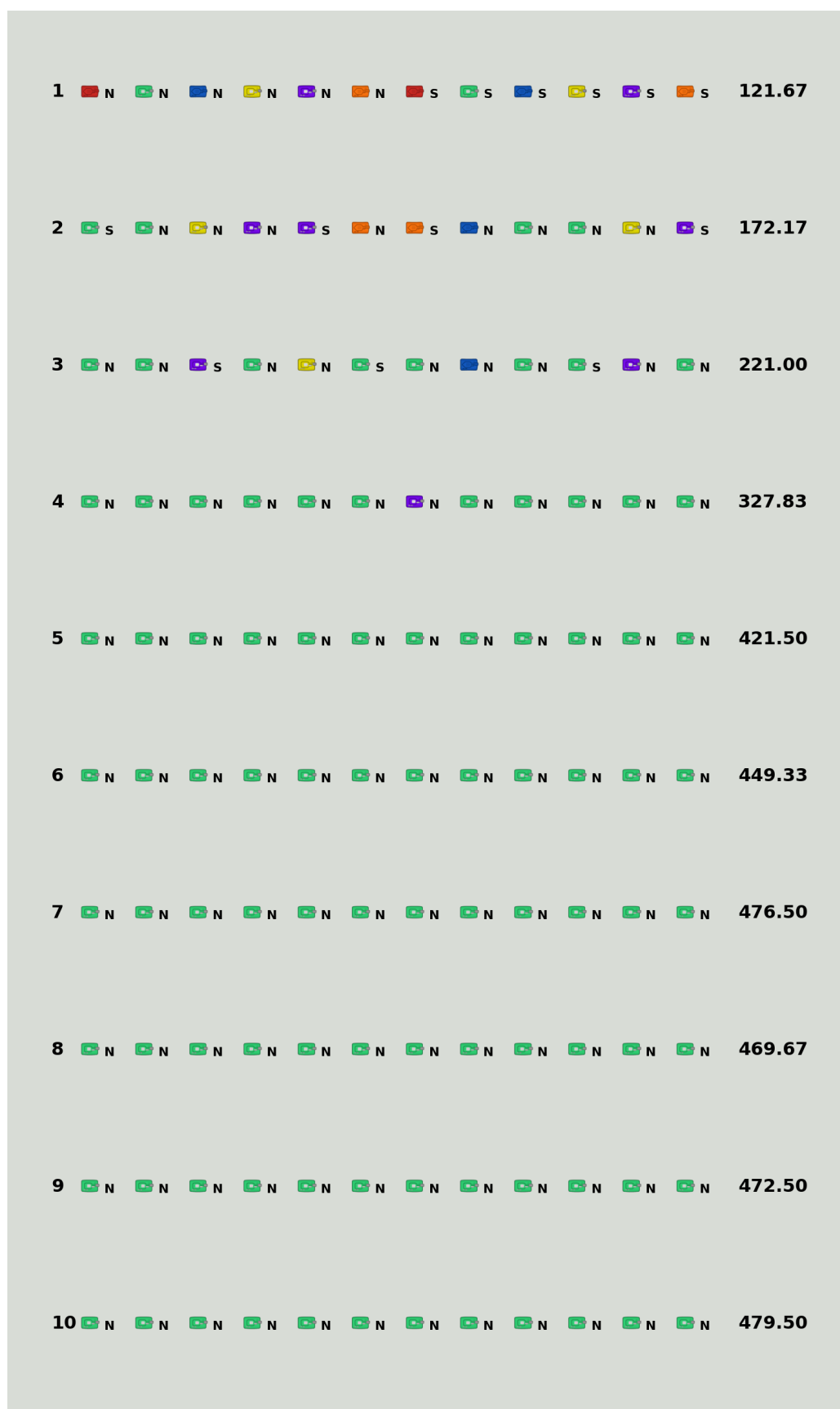


Figura C.1: Visualização da moda de cada onda com a versão v3 contra Torres Verdes.

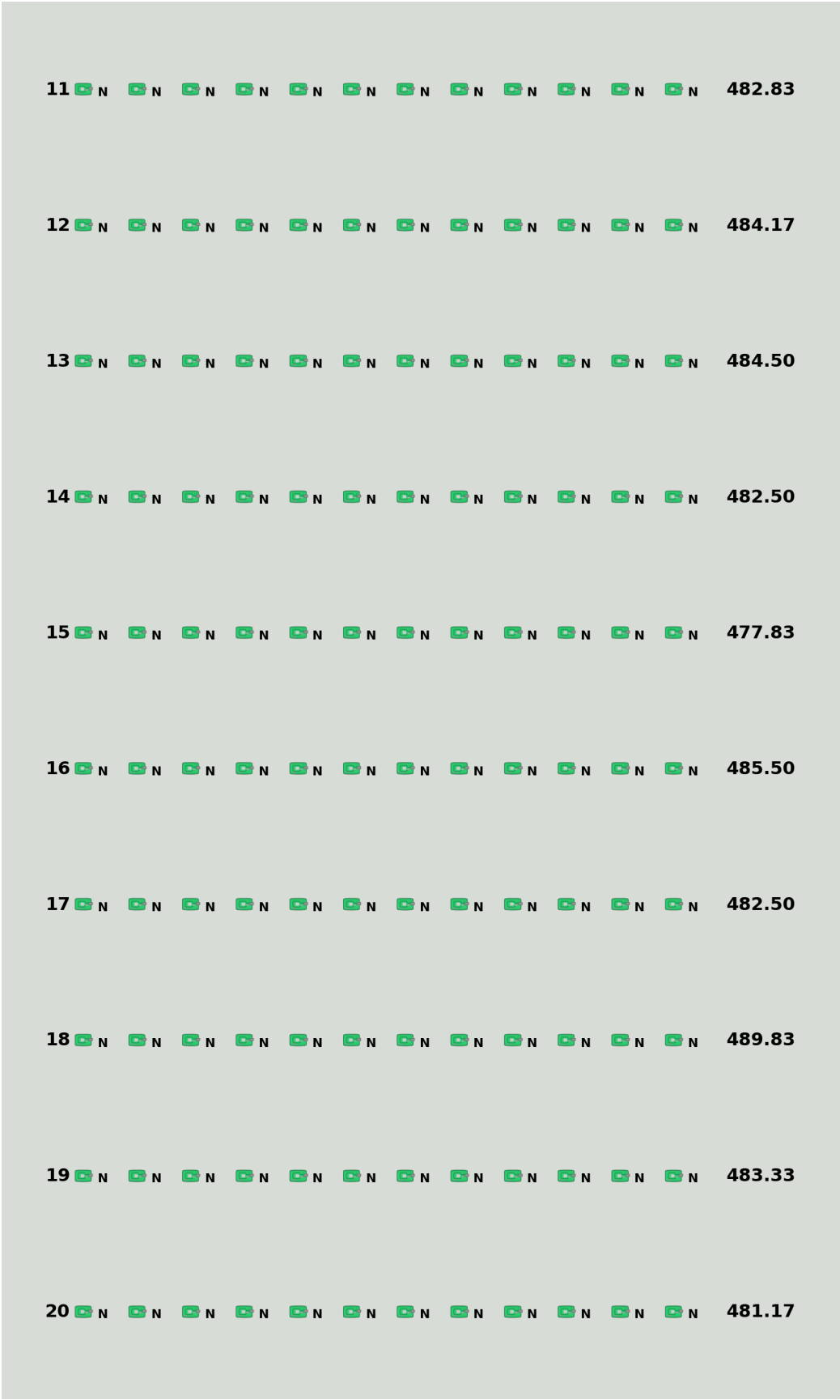


Figura C.2: Visualização da moda de cada onda com a versão v3 contra Torres Verdes.

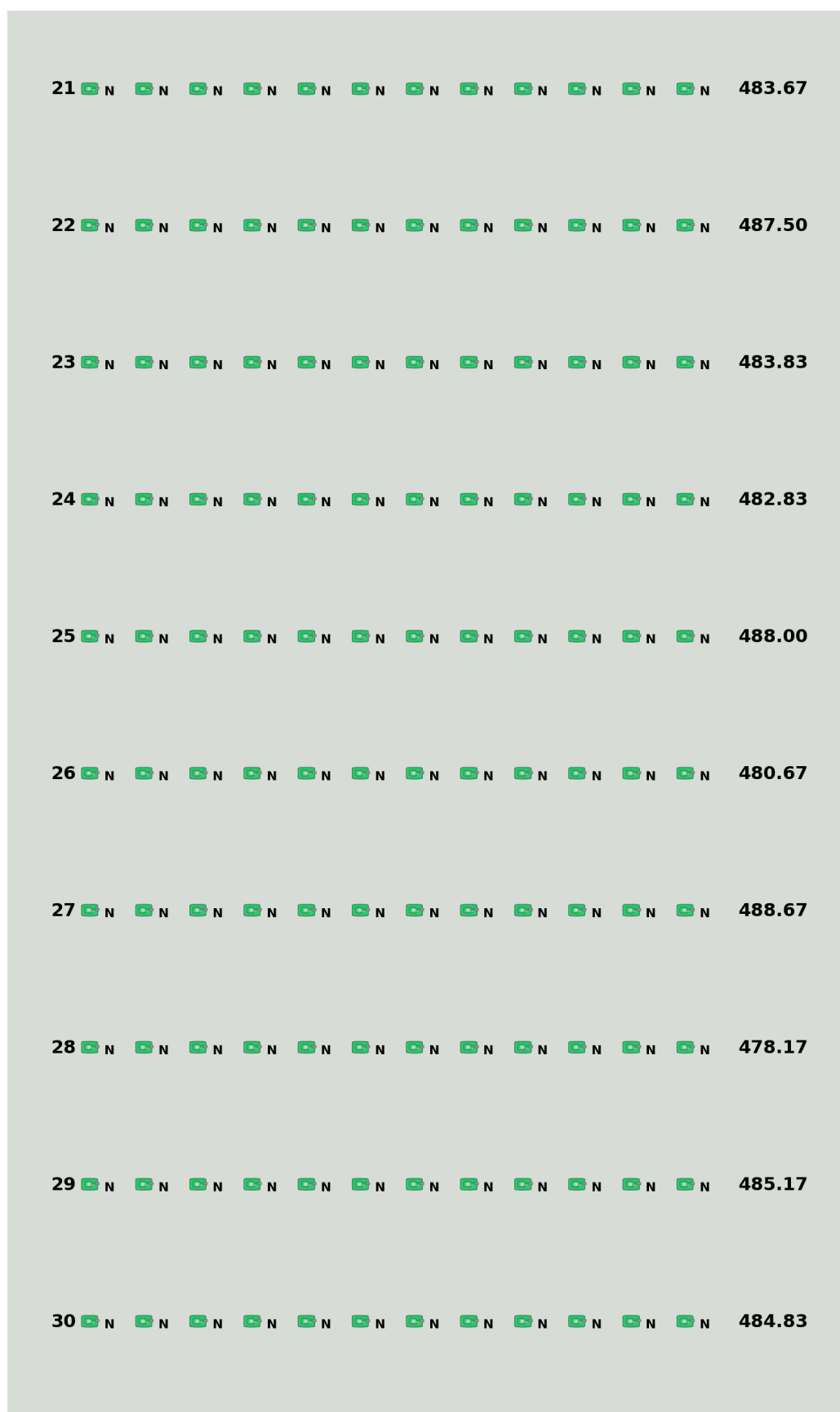


Figura C.3: Visualização da moda de cada onda com a versão v3 contra Torres Verdes.

C.2 Torres Vermelhas

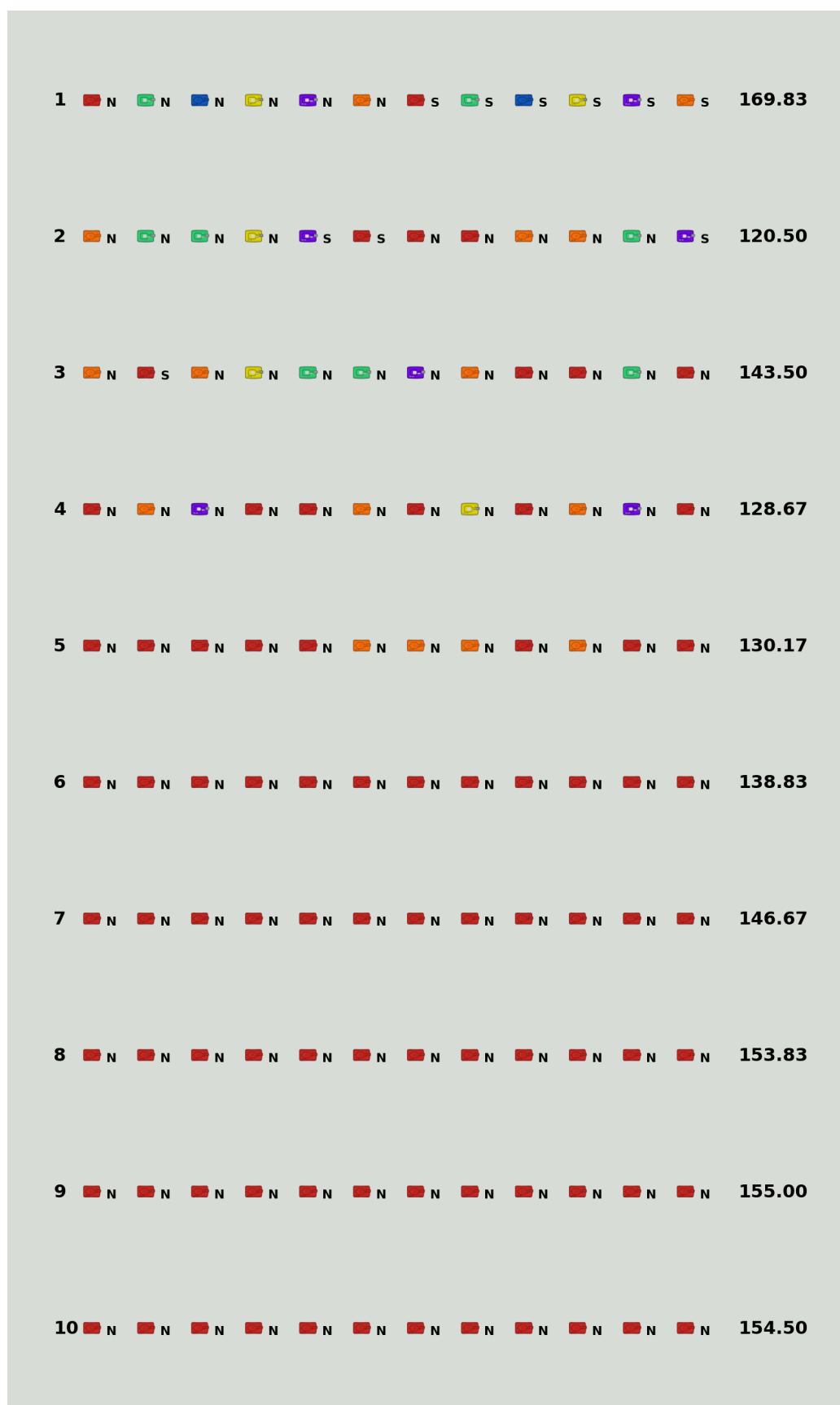


Figura C.4: Visualização da moda de cada onda com a versão v3 contra Torres Vermelhas.

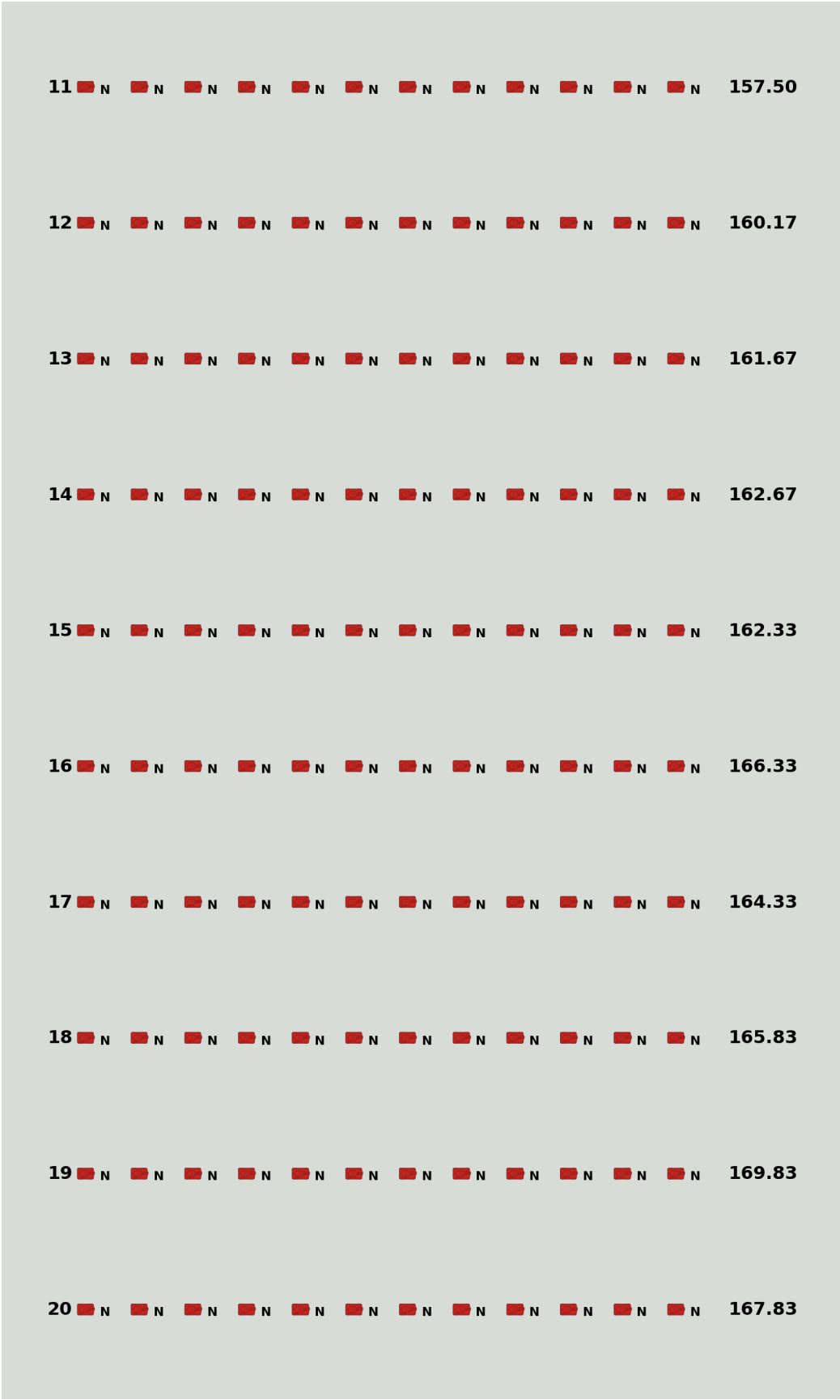


Figura C.5: Visualização da moda de cada onda com a versão v3 contra Torres Vermelhas.

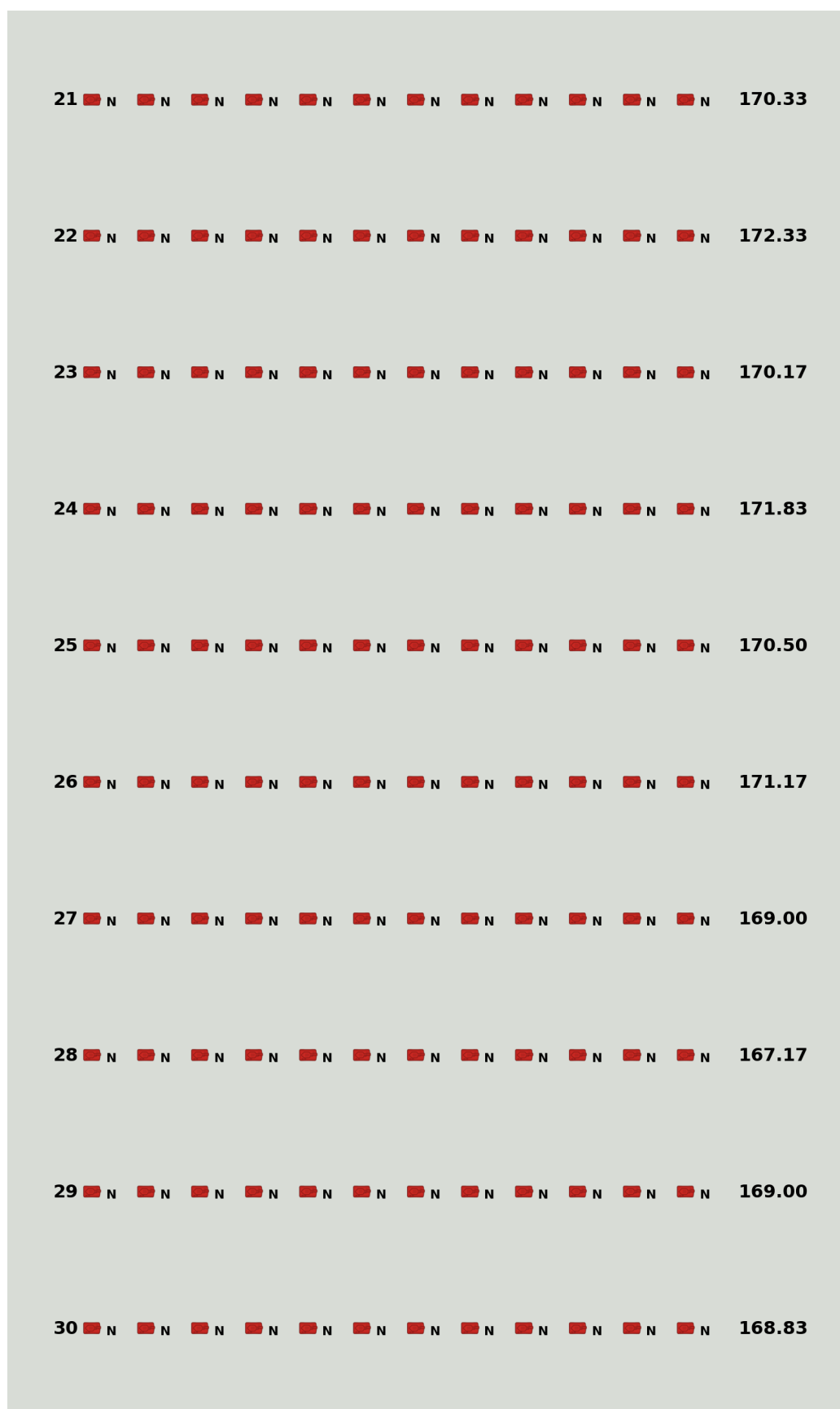


Figura C.6: Visualização da moda de cada onda com a versão v3 contra Torres Vermelhas.

C.3 Torres Verde + Vermelha

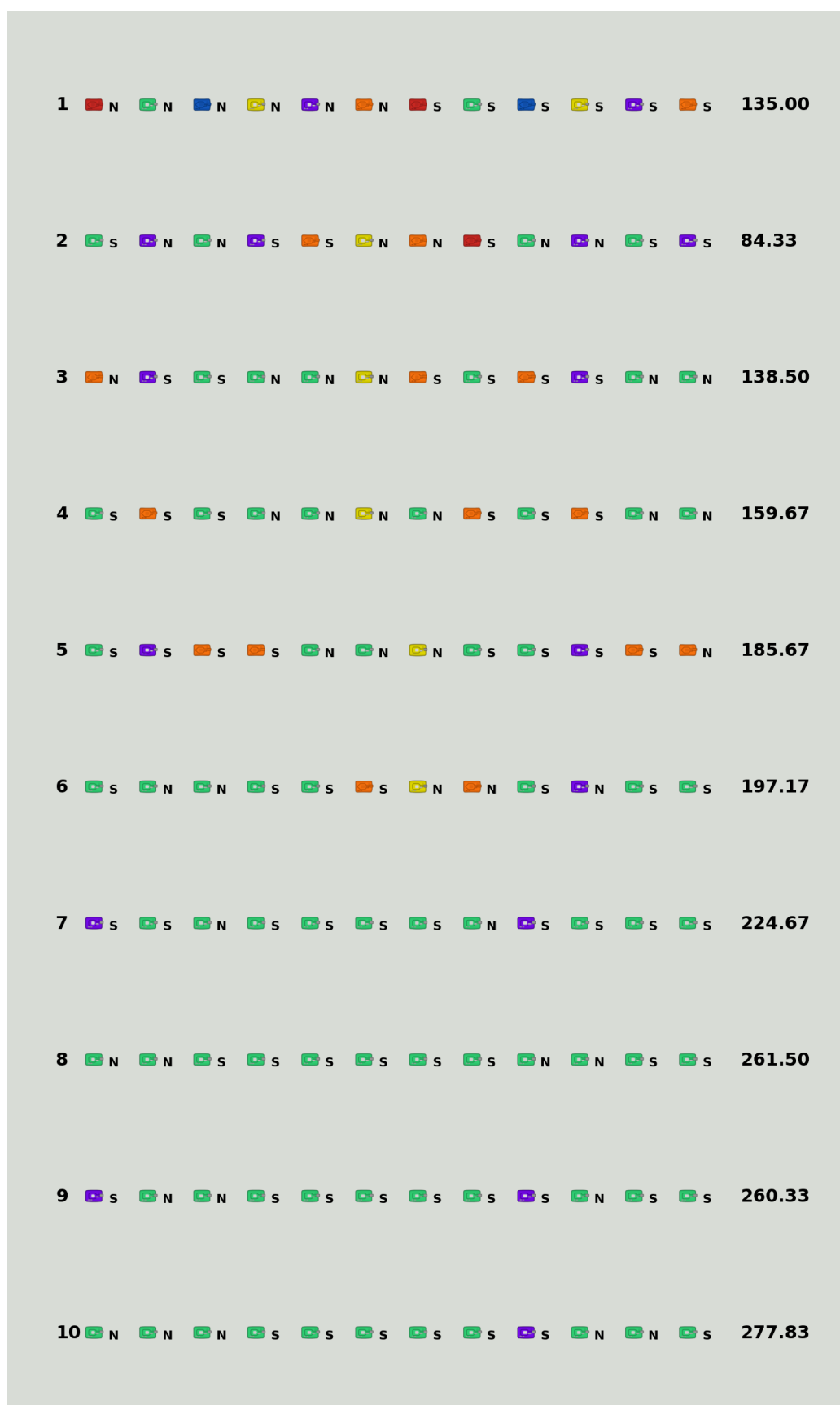


Figura C.7: Visualização da moda de cada onda com a versão v3 contra Torres Verdes + Vermelhas.

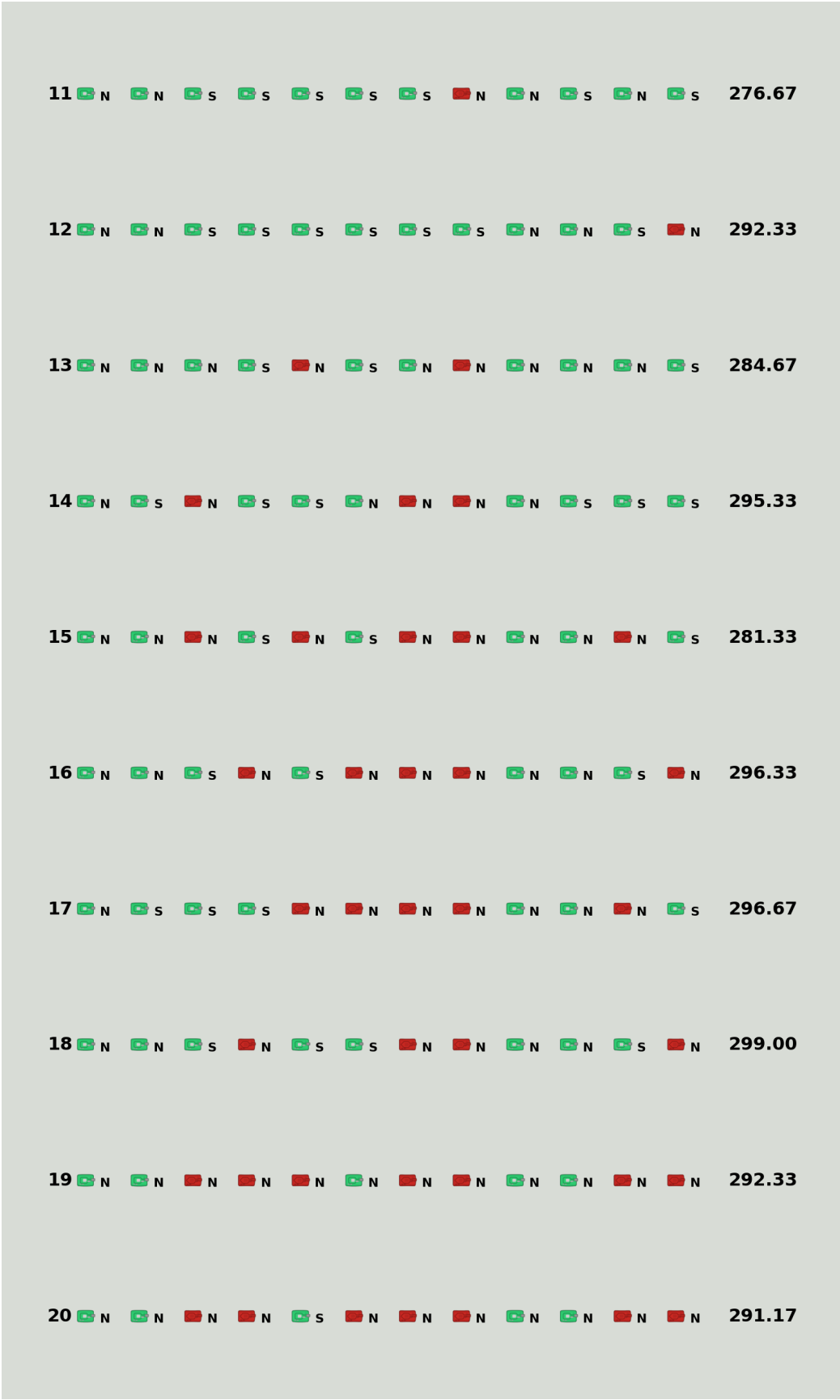


Figura C.8: Visualização da moda de cada onda com a versão v3 contra Torres Verdes + Vermelhas.

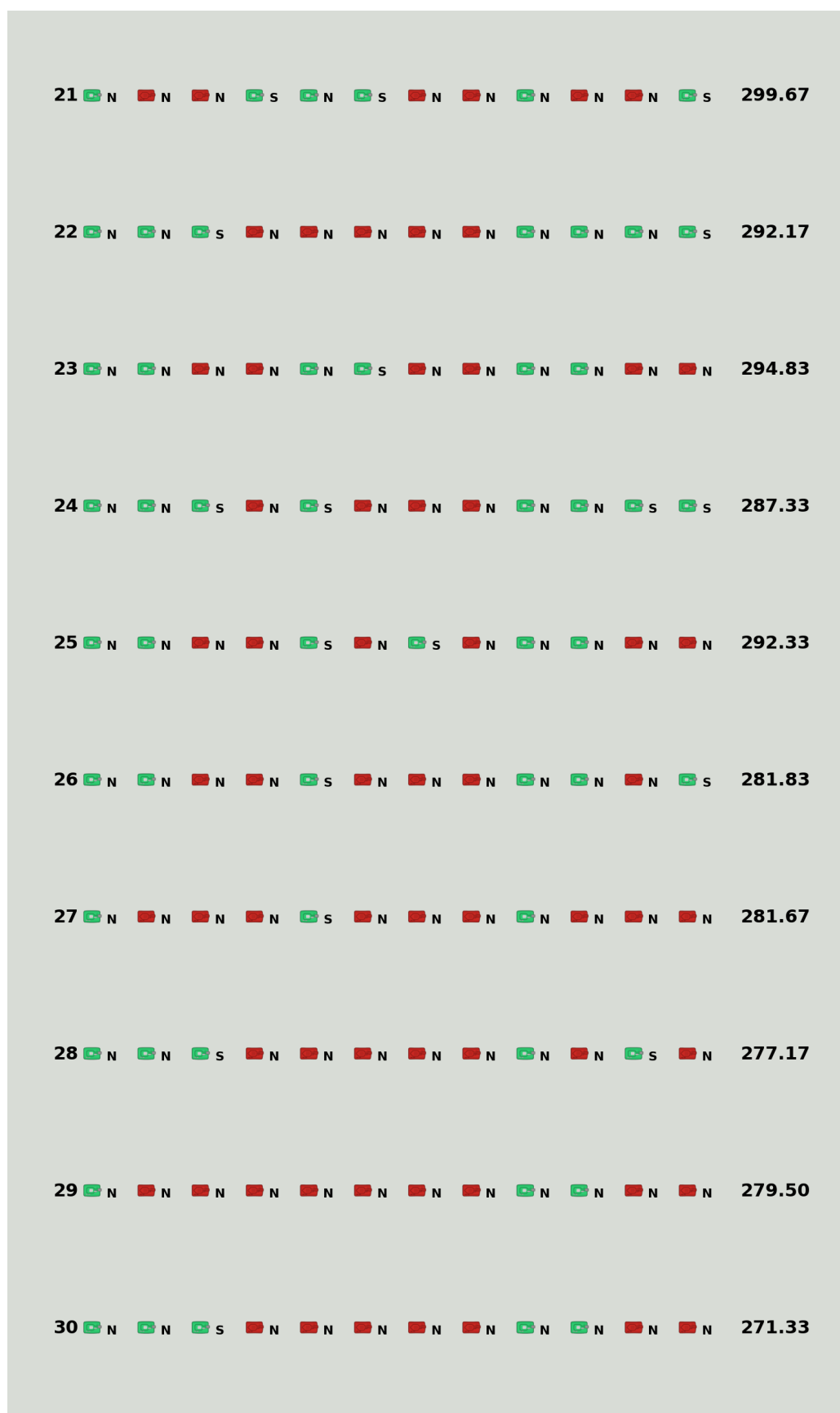


Figura C.9: Visualização da moda de cada onda com a versão v3 contra Torres Verdes + Vermelhas.

C.4 Torres Vermelha + Verde

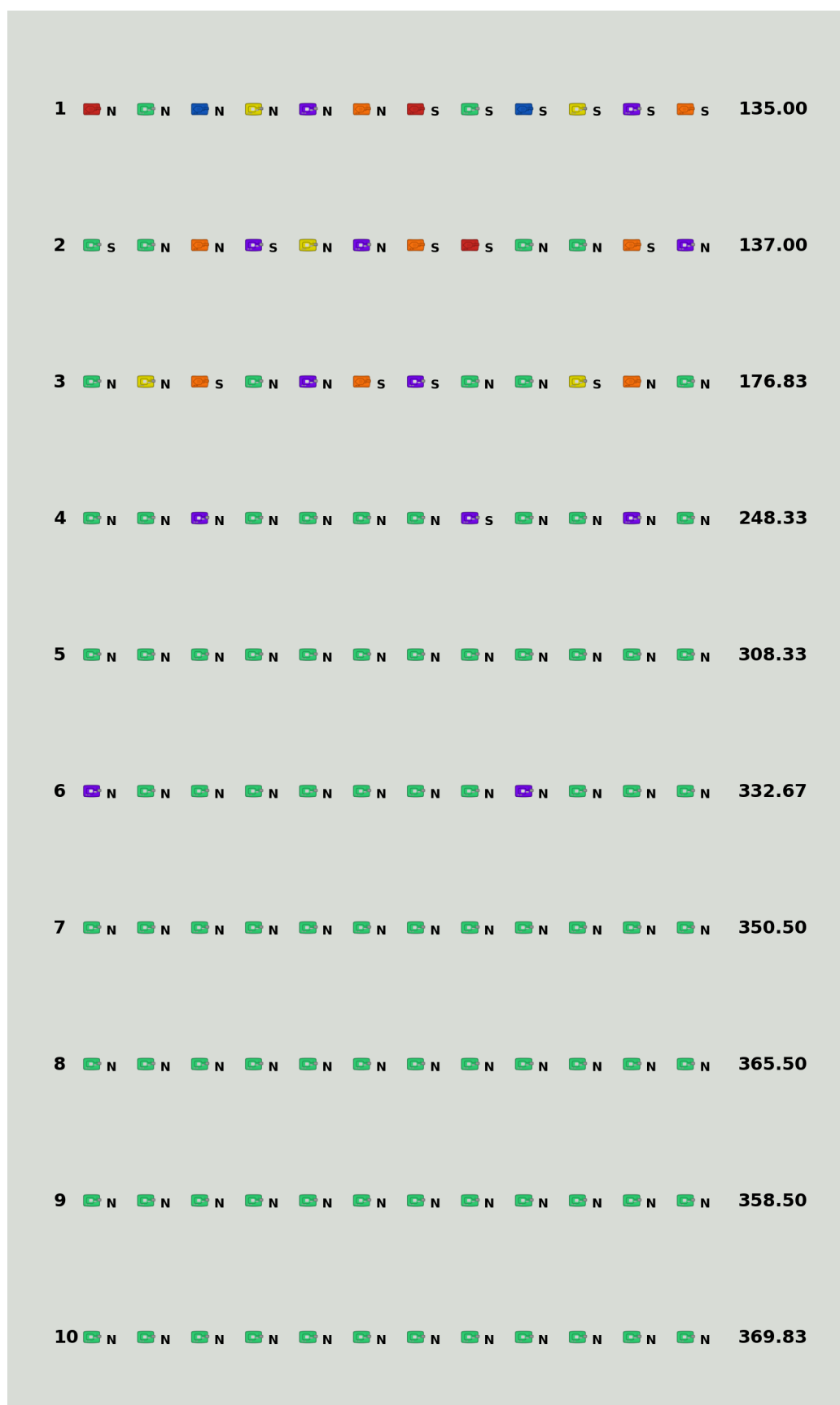


Figura C.10: Visualização da moda de cada onda com a versão v3 contra Torres Vermelhas + Verdes.

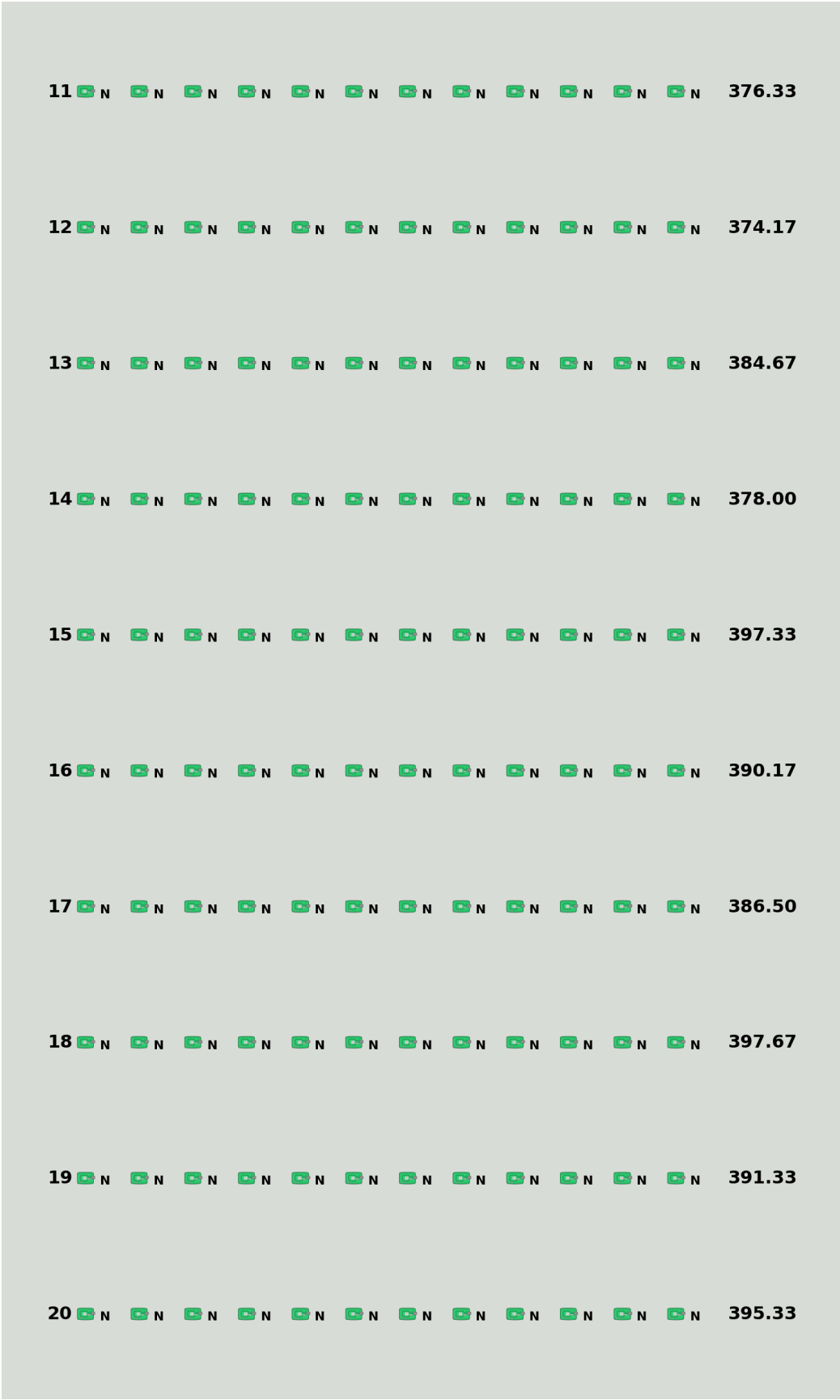


Figura C.11: Visualização da moda de cada onda com a versão v3 contra Torres Vermelhas + Verdes.

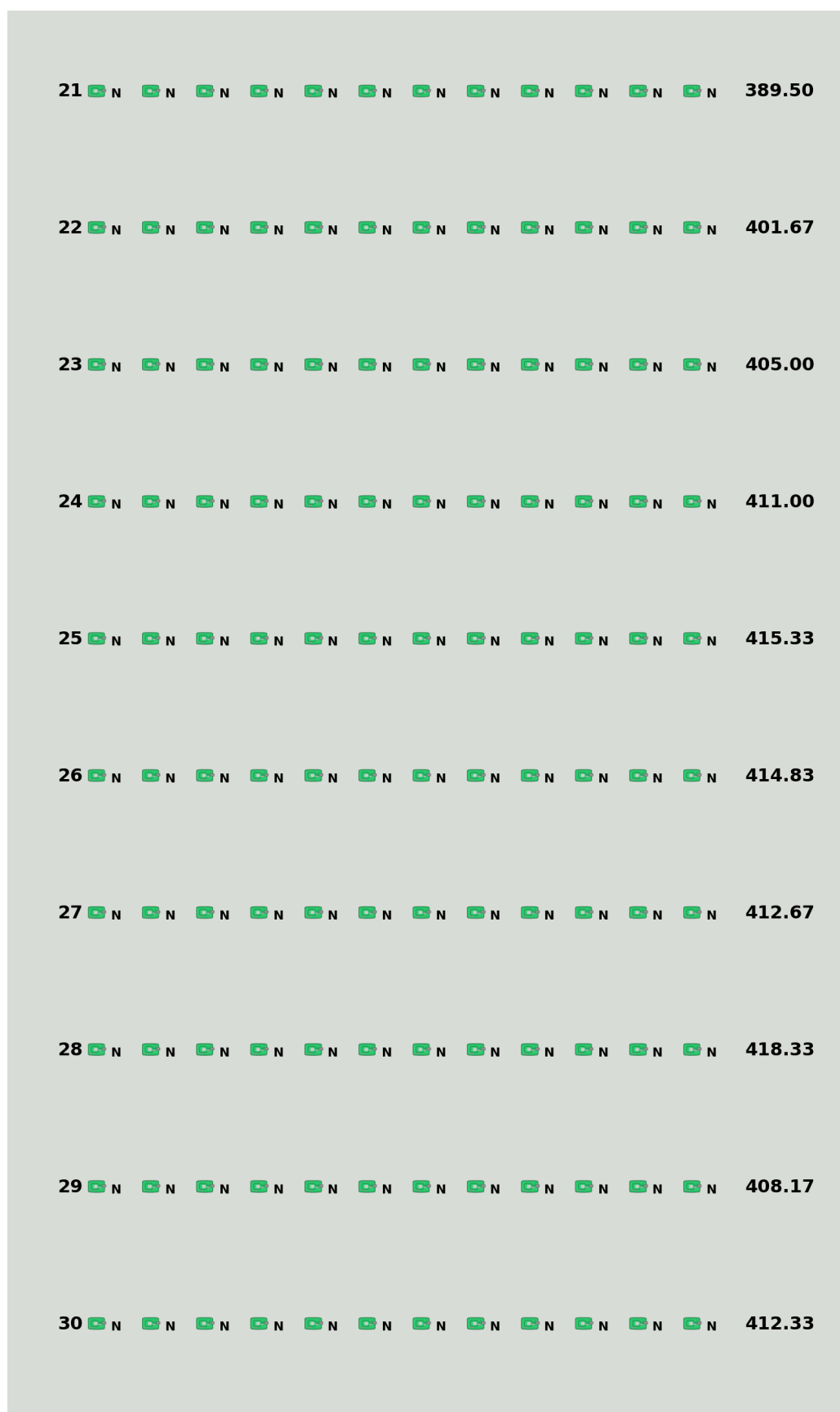


Figura C.12: Visualização da moda de cada onda com a versão v3 contra Torres Vermelhas + Verdes.

Apêndice D

Moda das Ondas no Space Shooter para versão v1

Foram calculadas as modas das ondas do *fitness* desenvolvido, para permitir a visualização dos inimigos mais comuns que o algoritmo convergiu.

D.1 Nave Parada com Disparo Amarelo

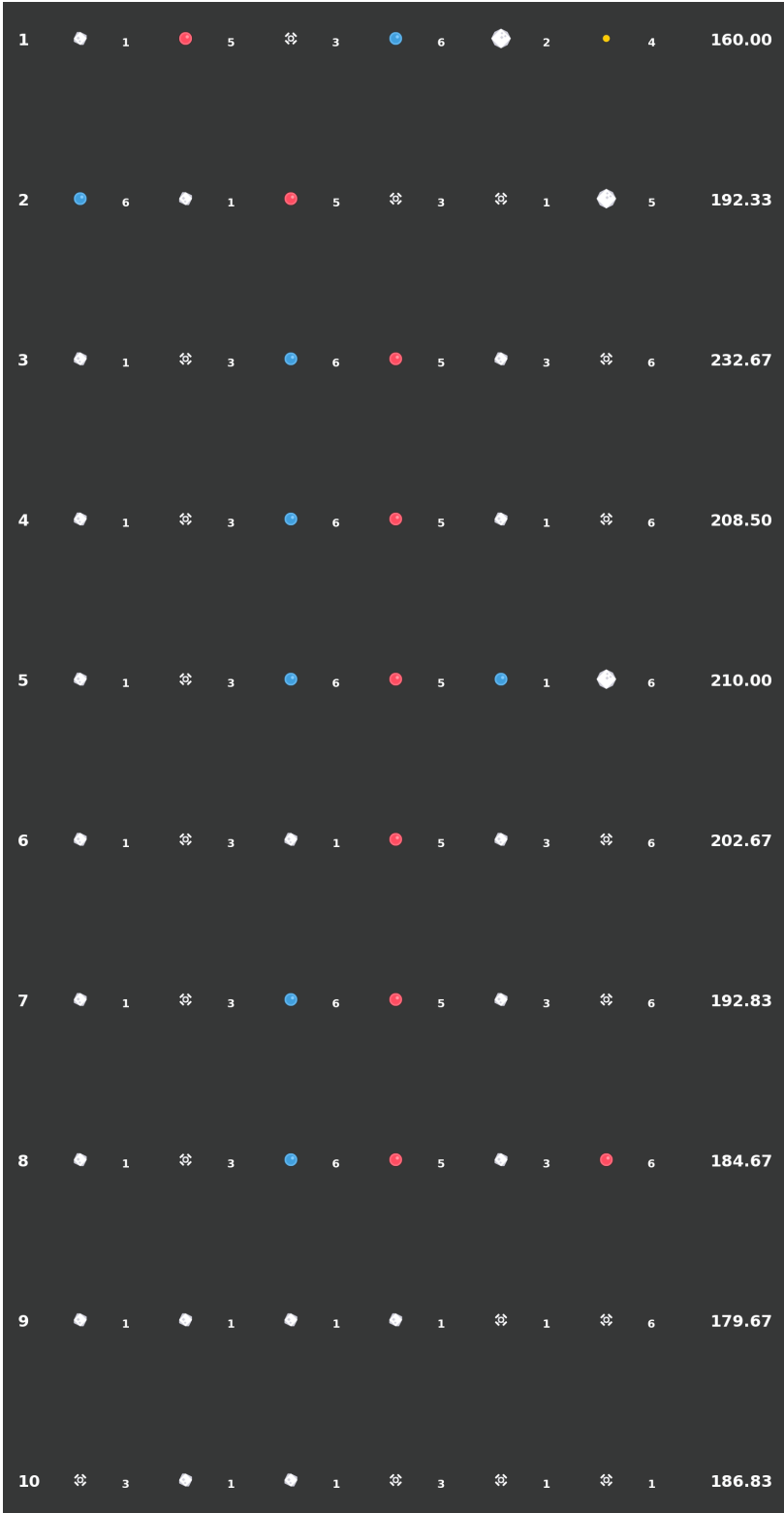


Figura D.1: Visualização da moda de cada onda com a versão v1 contra Nave Parada, Disparo Amarelo.



Figura D.2: Visualização da moda de cada onda com a versão v1 contra Nave Parada, Disparo Amarelo.

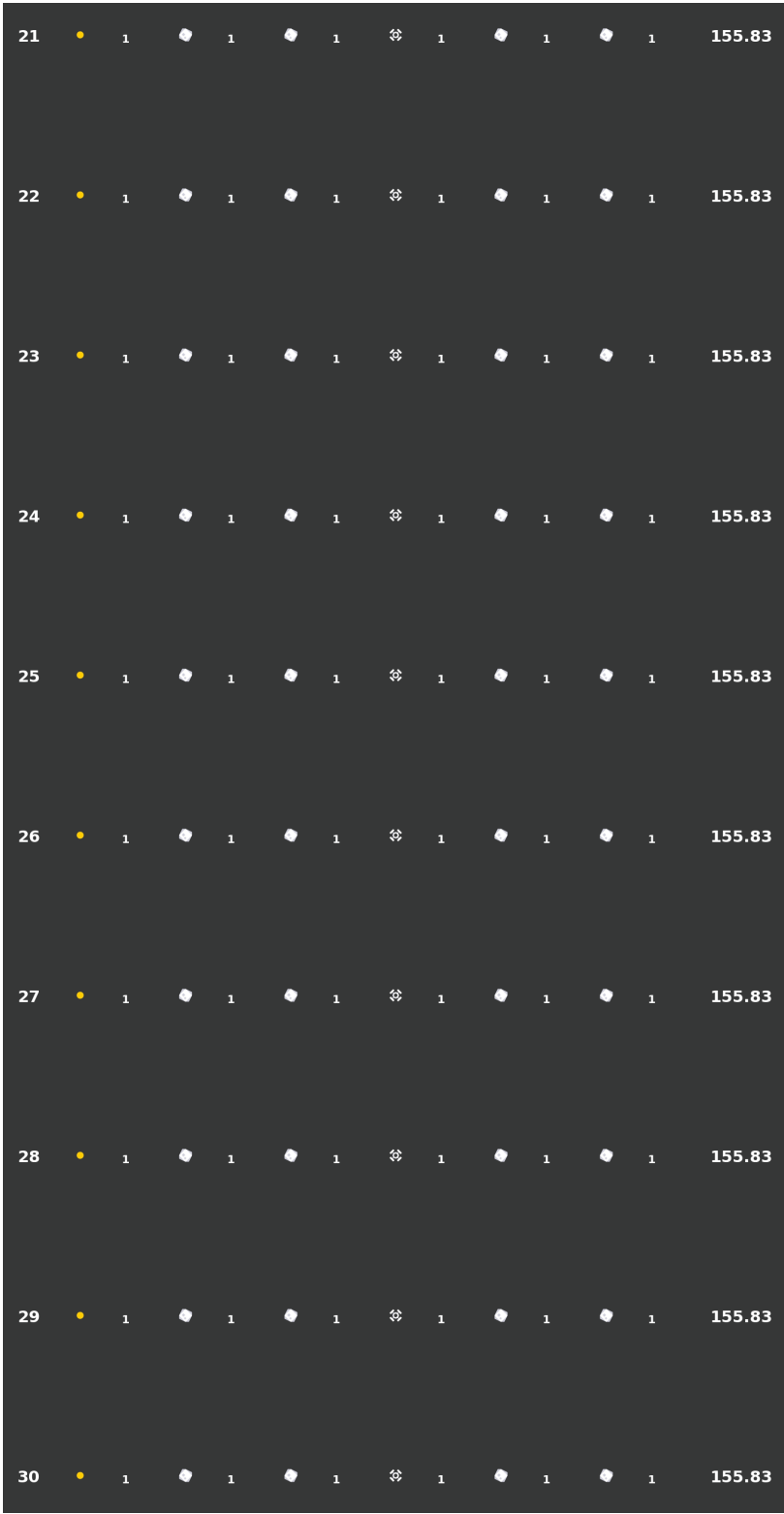


Figura D.3: Visualização da moda de cada onda com a versão v1 contra Nave Parada, Disparo Amarelo.

D.2 Nave Movendo com Disparo Amarelo

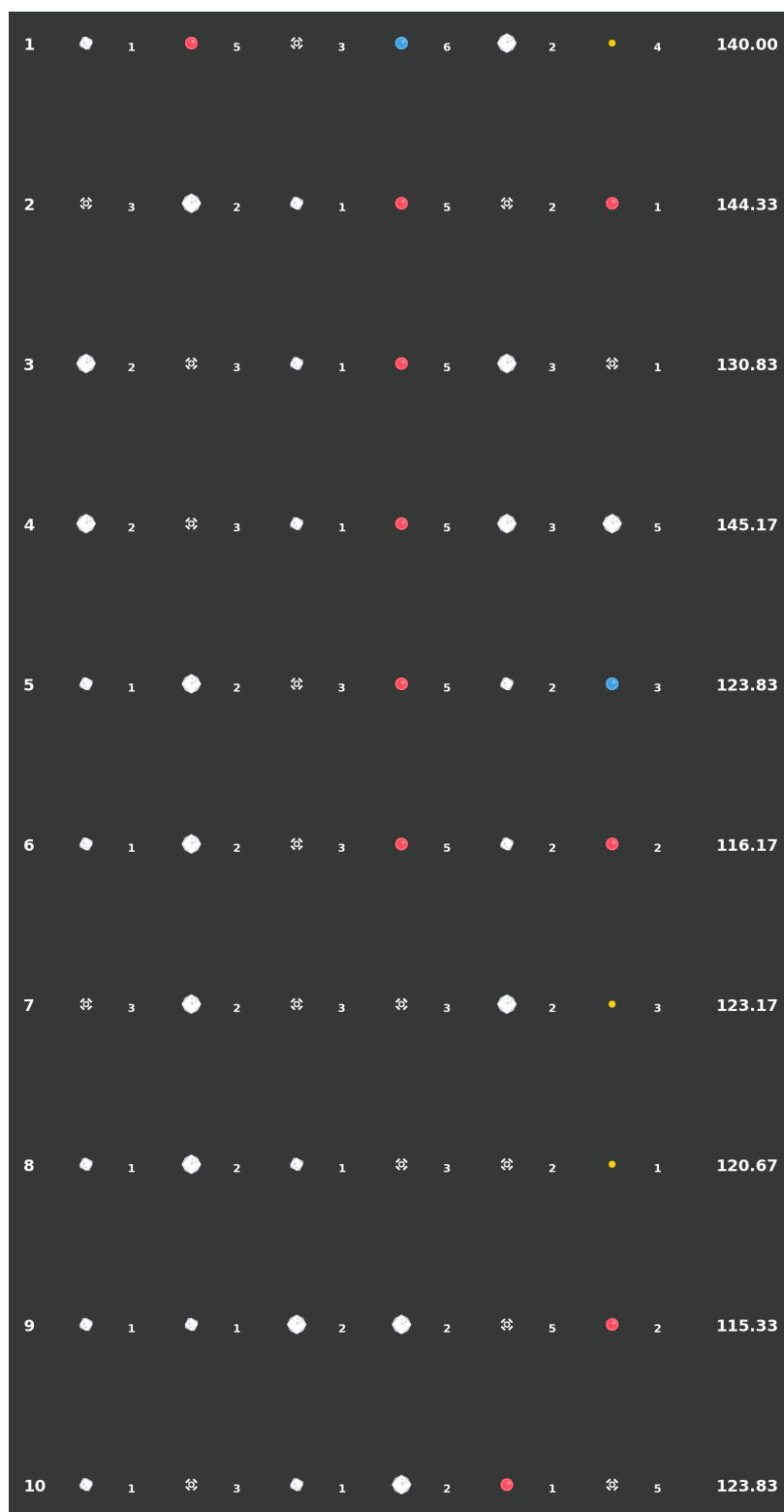


Figura D.4: Visualização da moda de cada onda com a versão v1 contra Nave Movendo, Disparo Amarelo.

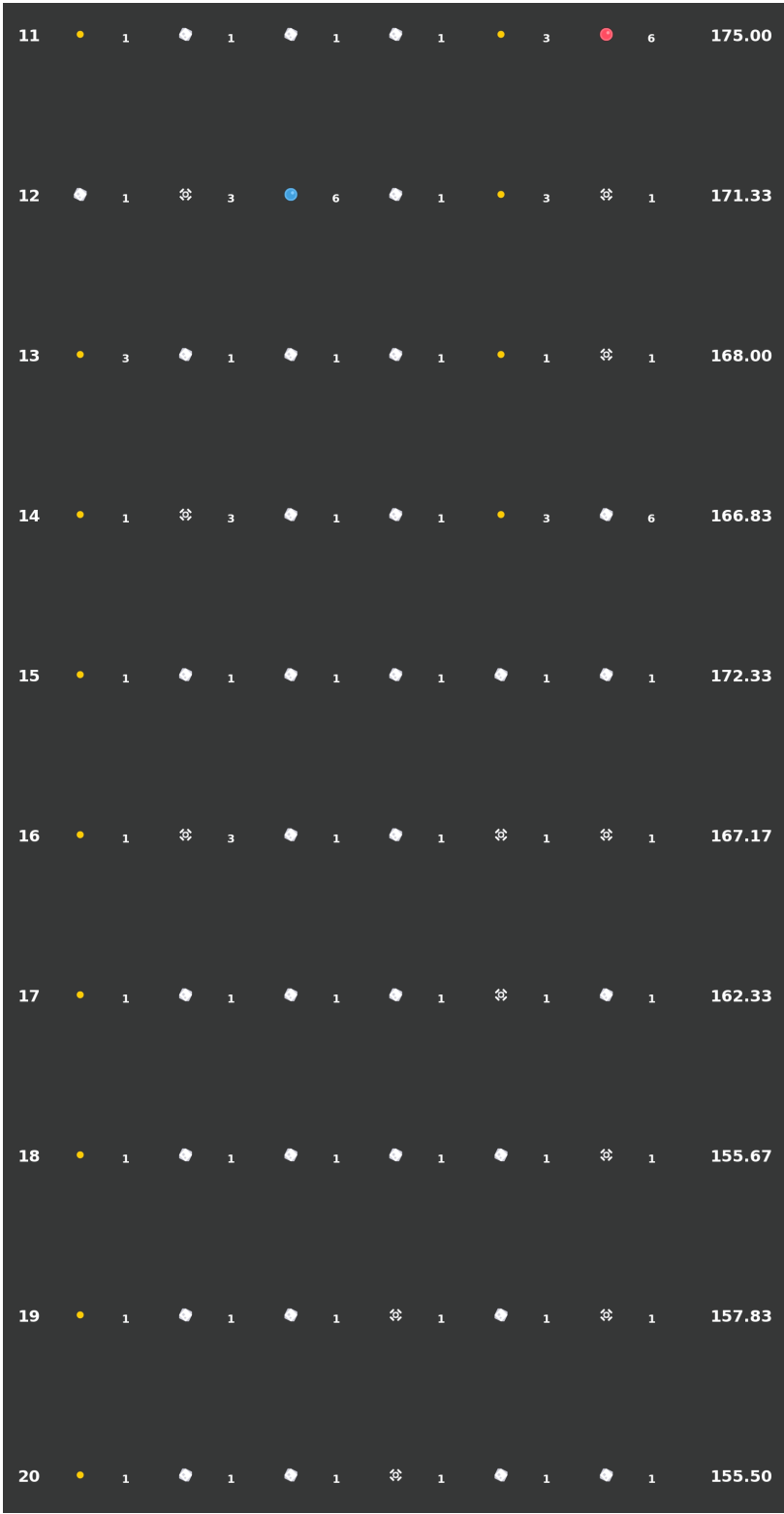


Figura D.5: *Visualização da moda de cada onda com a versão v1 contra Nave Movendo, Disparo Amarelo.*

21	●	1	◆	1	◆	1	⊗	1	◆	1	◆	1	155.83
22	●	1	◆	1	◆	1	⊗	1	◆	1	◆	1	155.83
23	●	1	◆	1	◆	1	⊗	1	◆	1	◆	1	155.83
24	●	1	◆	1	◆	1	⊗	1	◆	1	◆	1	155.83
25	●	1	◆	1	◆	1	⊗	1	◆	1	◆	1	155.83
26	●	1	◆	1	◆	1	⊗	1	◆	1	◆	1	155.83
27	●	1	◆	1	◆	1	⊗	1	◆	1	◆	1	155.83
28	●	1	◆	1	◆	1	⊗	1	◆	1	◆	1	155.83
29	●	1	◆	1	◆	1	⊗	1	◆	1	◆	1	155.83
30	●	1	◆	1	◆	1	⊗	1	◆	1	◆	1	155.83

Figura D.6: Visualização da moda de cada onda com a versão v1 contra Nave Movendo, Disparo Amarelo.

D.3 Nave Parada com Disparo Vermelho

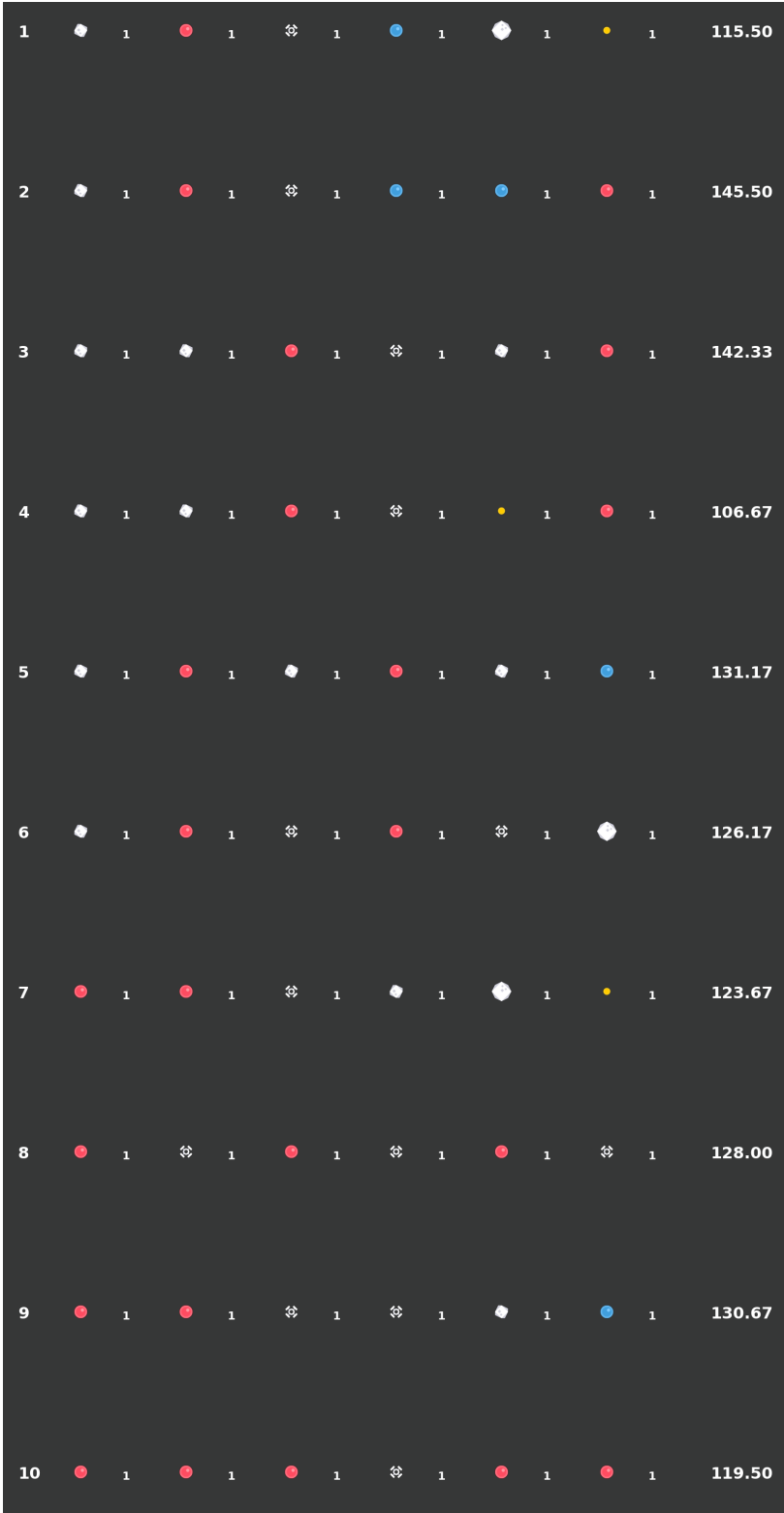


Figura D.7: Visualização da moda de cada onda com a versão v1 contra Nave Parada, Disparo Vermelho.

11	●	1	●	1	⊗	1	●	1	●	1	⊗	1	120.83
12	●	1	●	1	●	1	●	1	●	1	●	1	122.00
13	●	1	●	1	⊗	1	●	1	⊗	1	●	1	117.83
14	●	1	●	1	●	1	⊗	1	●	1	⊗	1	112.83
15	●	1	●	1	●	1	●	1	●	1	●	1	106.00
16	●	1	⊗	1	●	1	⊗	1	●	1	⊗	1	115.00
17	●	1	⊗	1	●	1	⊗	1	●	1	●	1	112.50
18	●	1	●	1	●	1	⊗	1	●	1	●	1	111.50
19	●	1	●	1	●	1	⊗	1	●	1	●	1	110.50
20	●	1	●	1	●	1	⊗	1	●	1	●	1	95.00

Figura D.8: Visualização da moda de cada onda com a versão v1 contra Nave Parada, Disparo Vermelho.



Figura D.9: Visualização da moda de cada onda com a versão v1 contra Nave Parada, Disparo Vermelho.

D.4 Nave Movendo com Disparo Vermelho

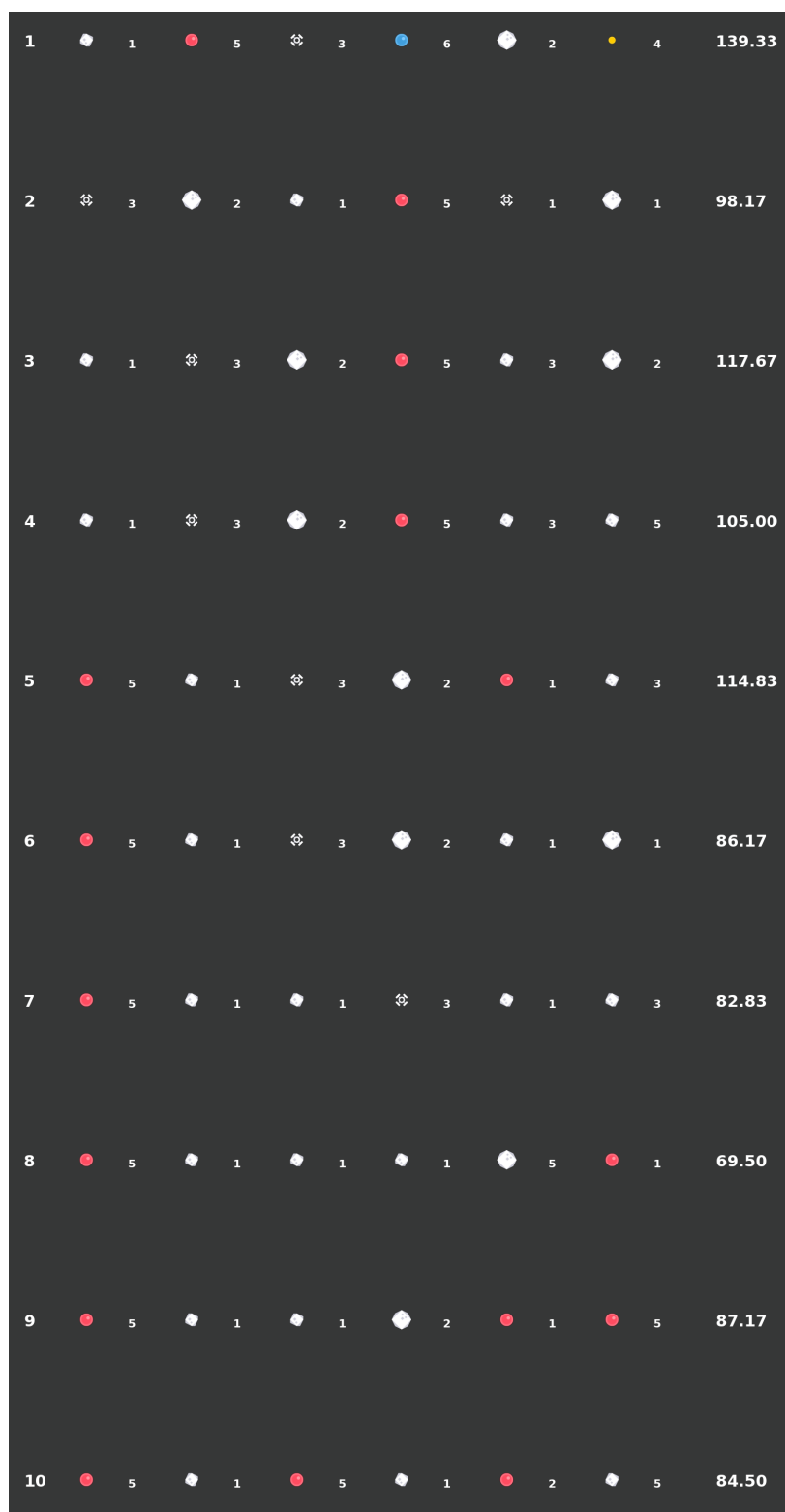


Figura D.10: Visualização da moda de cada onda com a versão v1 contra Nave Movendo, Disparo Vermelho.

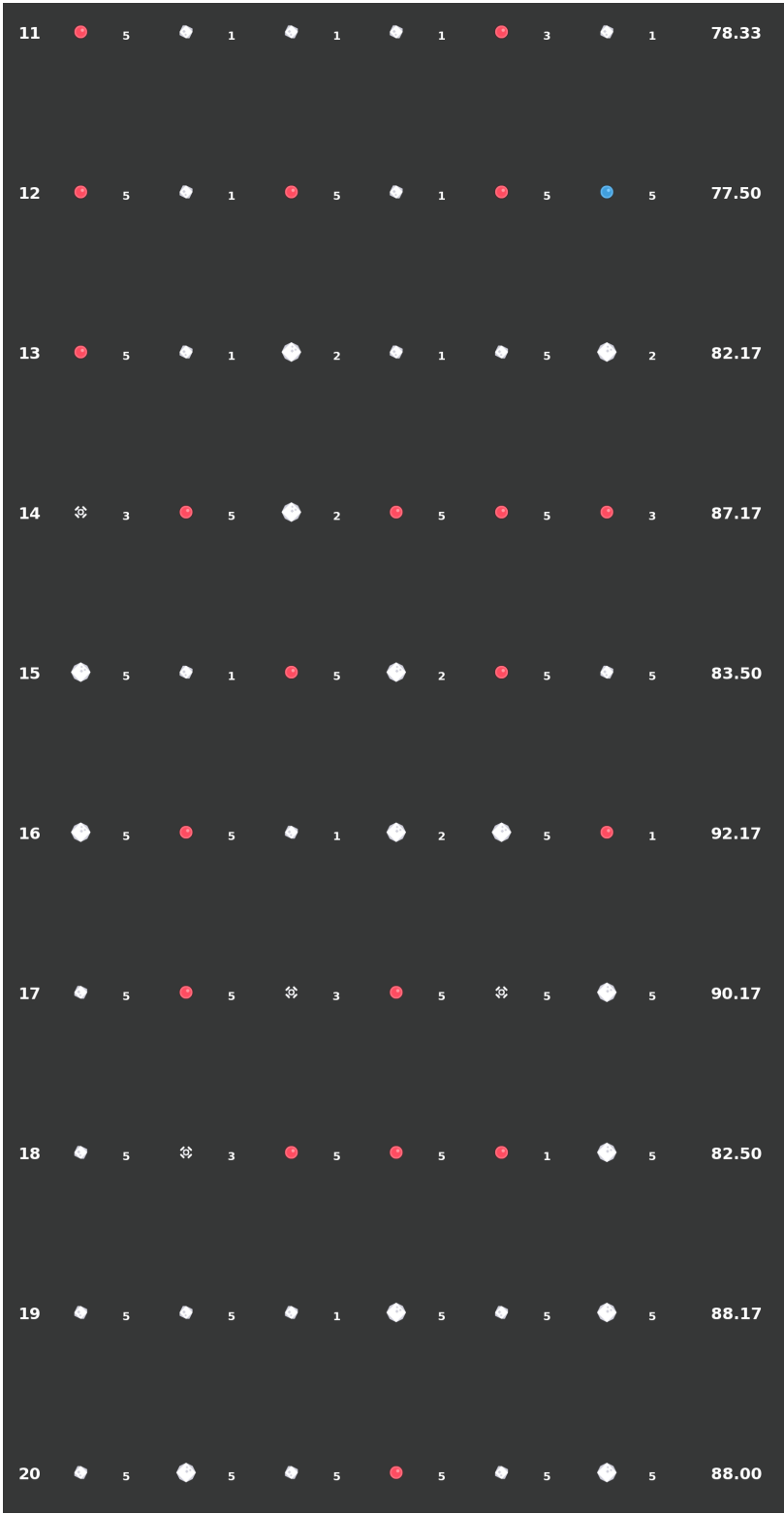


Figura D.11: Visualização da moda de cada onda com a versão v1 contra Nave Movendo, Disparo Vermelho.

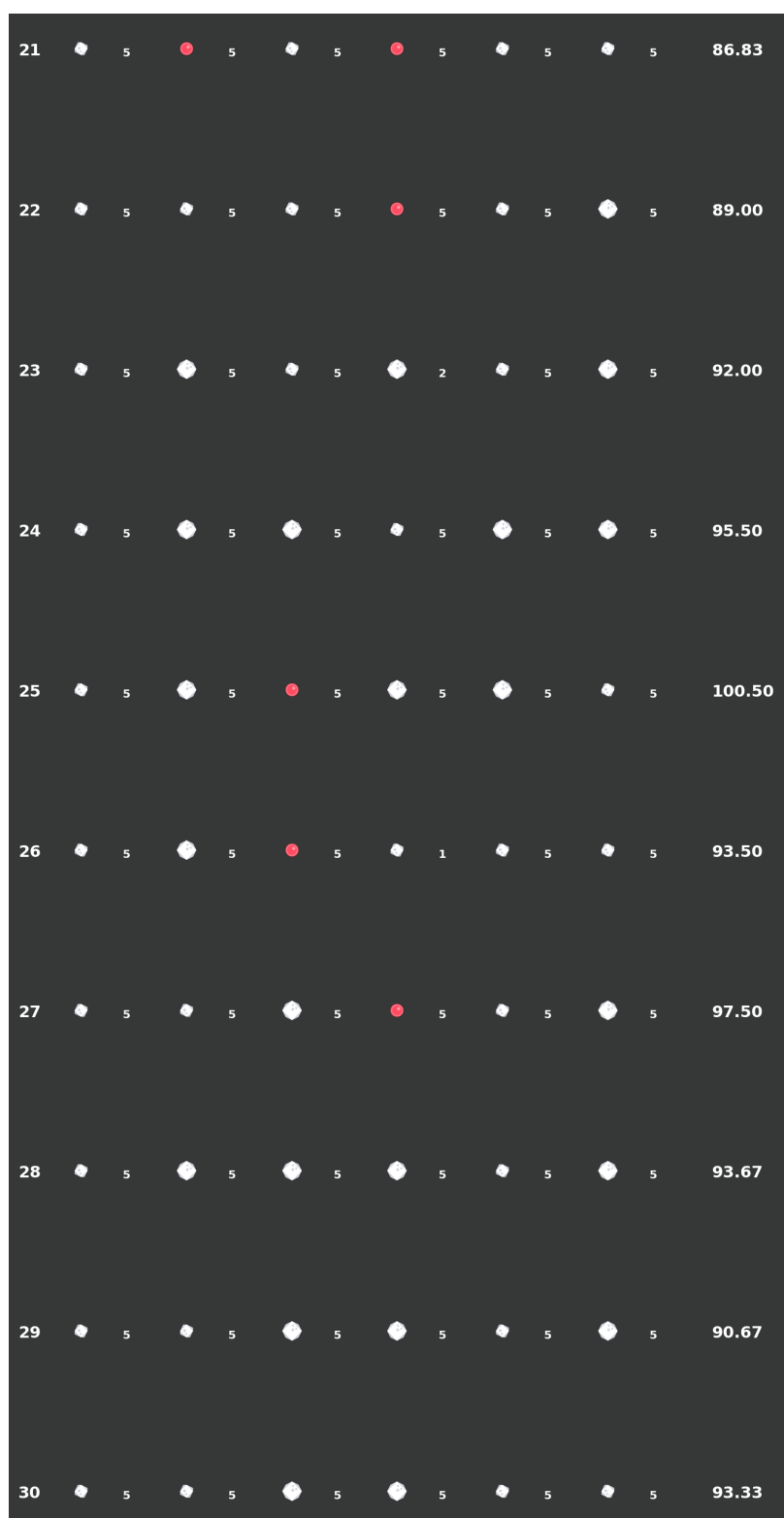


Figura D.12: Visualização da moda de cada onda com a versão v1 contra Nave Movendo, Disparo Vermelho.

Apêndice E

Moda das Ondas no Space Shooter para versão v3

Foram calculadas as modas das ondas do *fitness* desenvolvido, para permitir a visualização dos inimigos mais comuns que o algoritmo convergiu.

E.1 Nave Parada com Disparo Amarelo

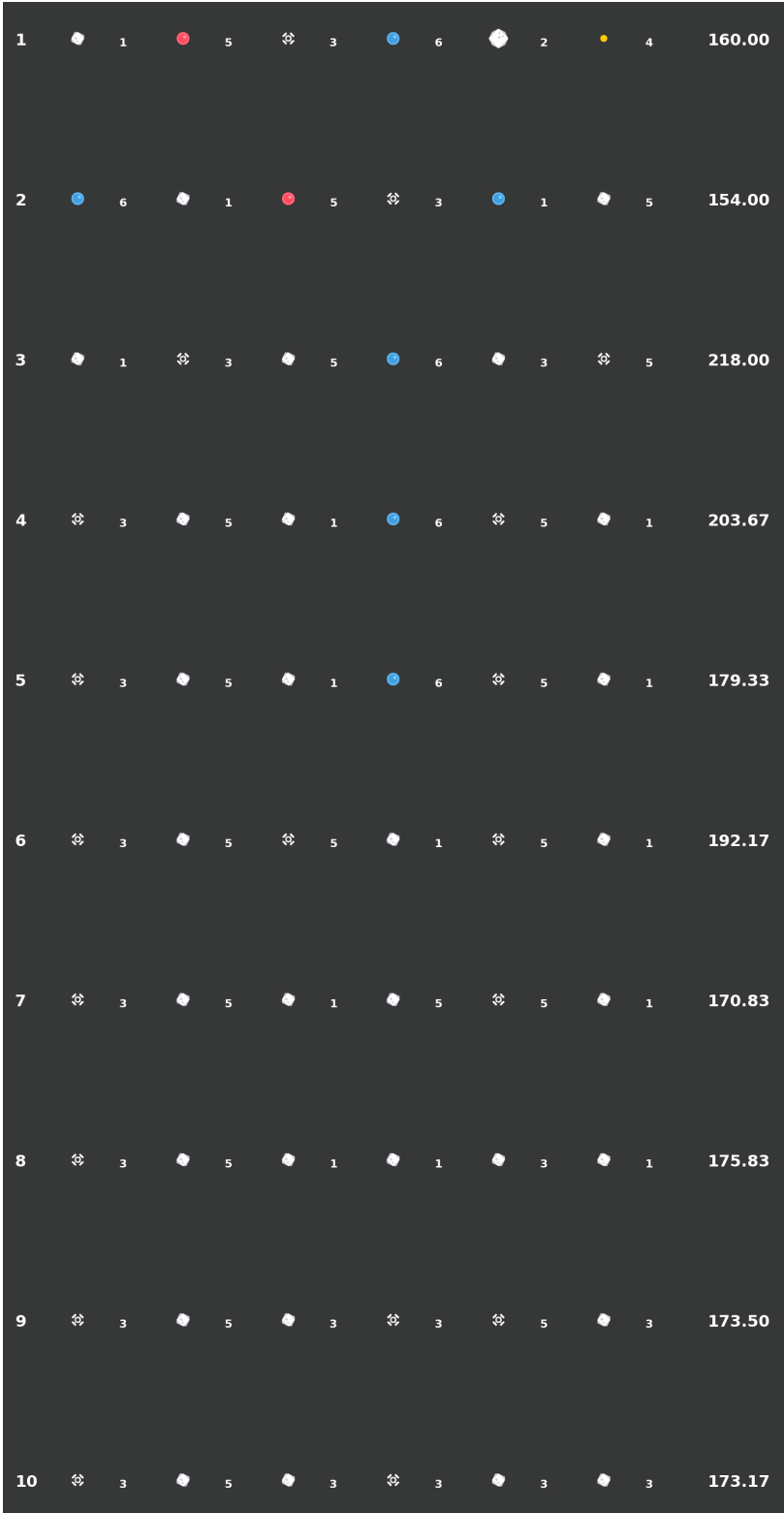


Figura E.1: Visualização da moda de cada onda com o fitness v3 contra Nave Parada, Disparo Amarelo.

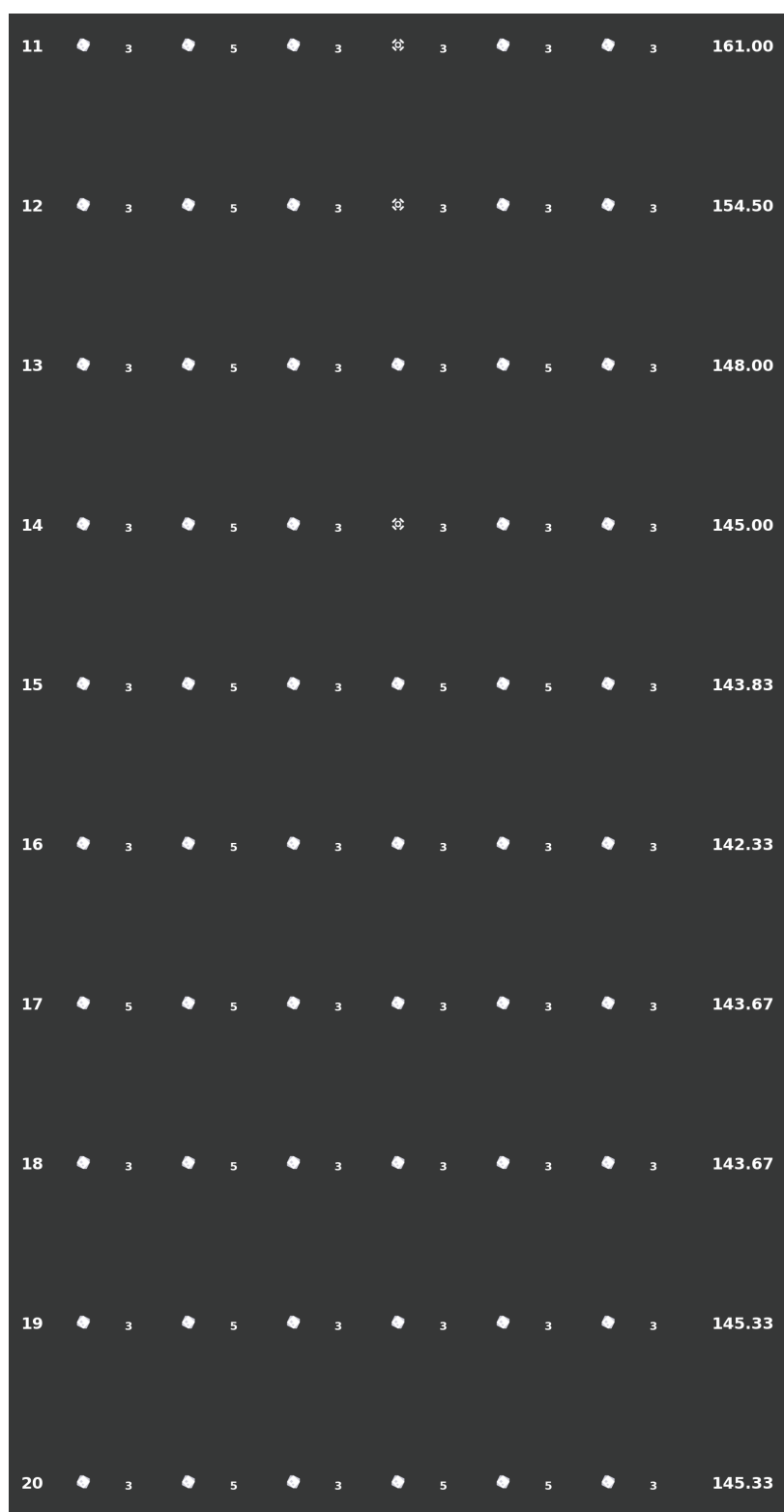


Figura E.2: Visualização da moda de cada onda com o fitness v3 contra Nave Parada, Disparo Amarelo.

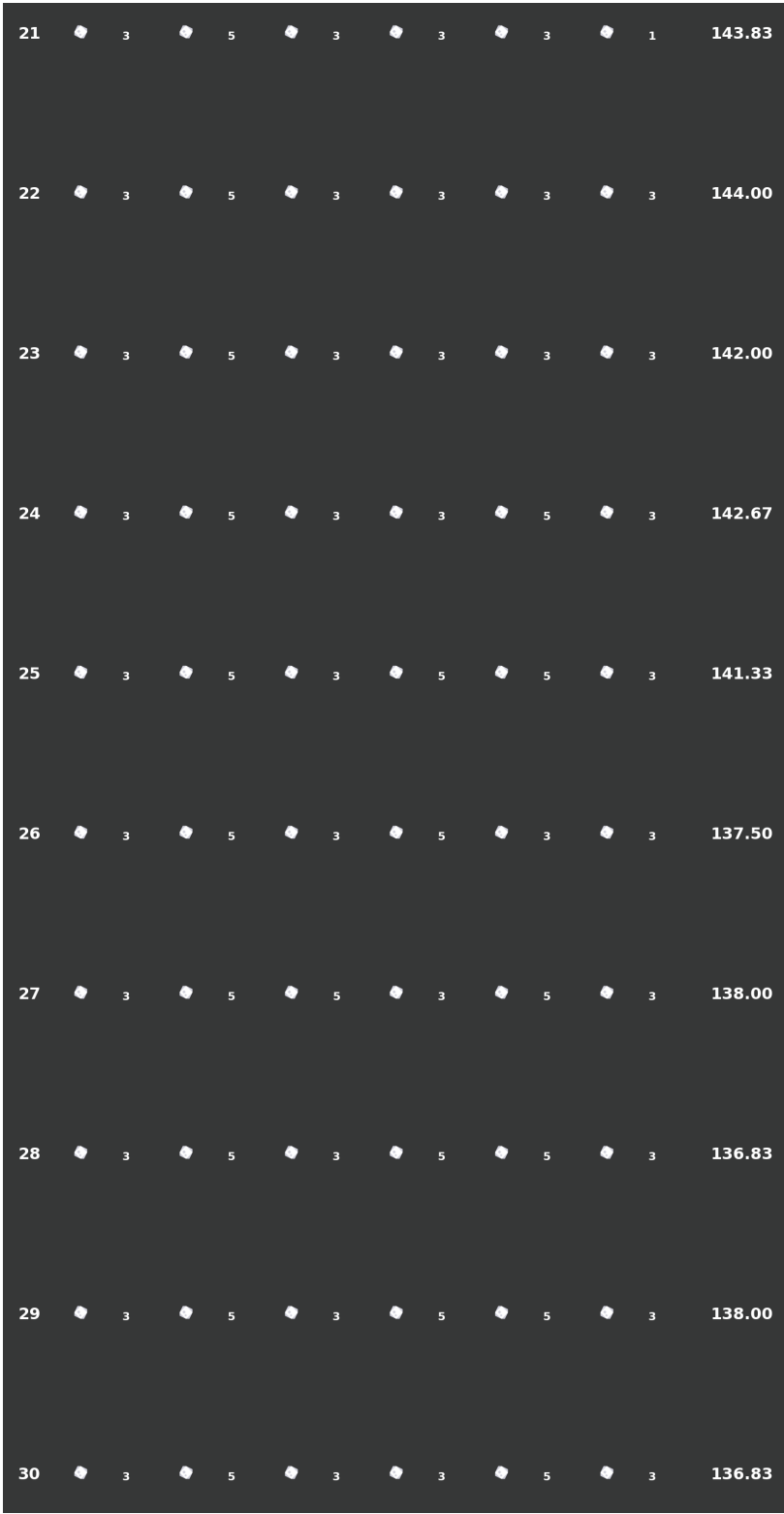


Figura E.3: *Visualização da moda de cada onda com o fitness v3 contra Nave Parada, Disparo Amarelo.*

E.2 Nave Movendo com Disparo Amarelo

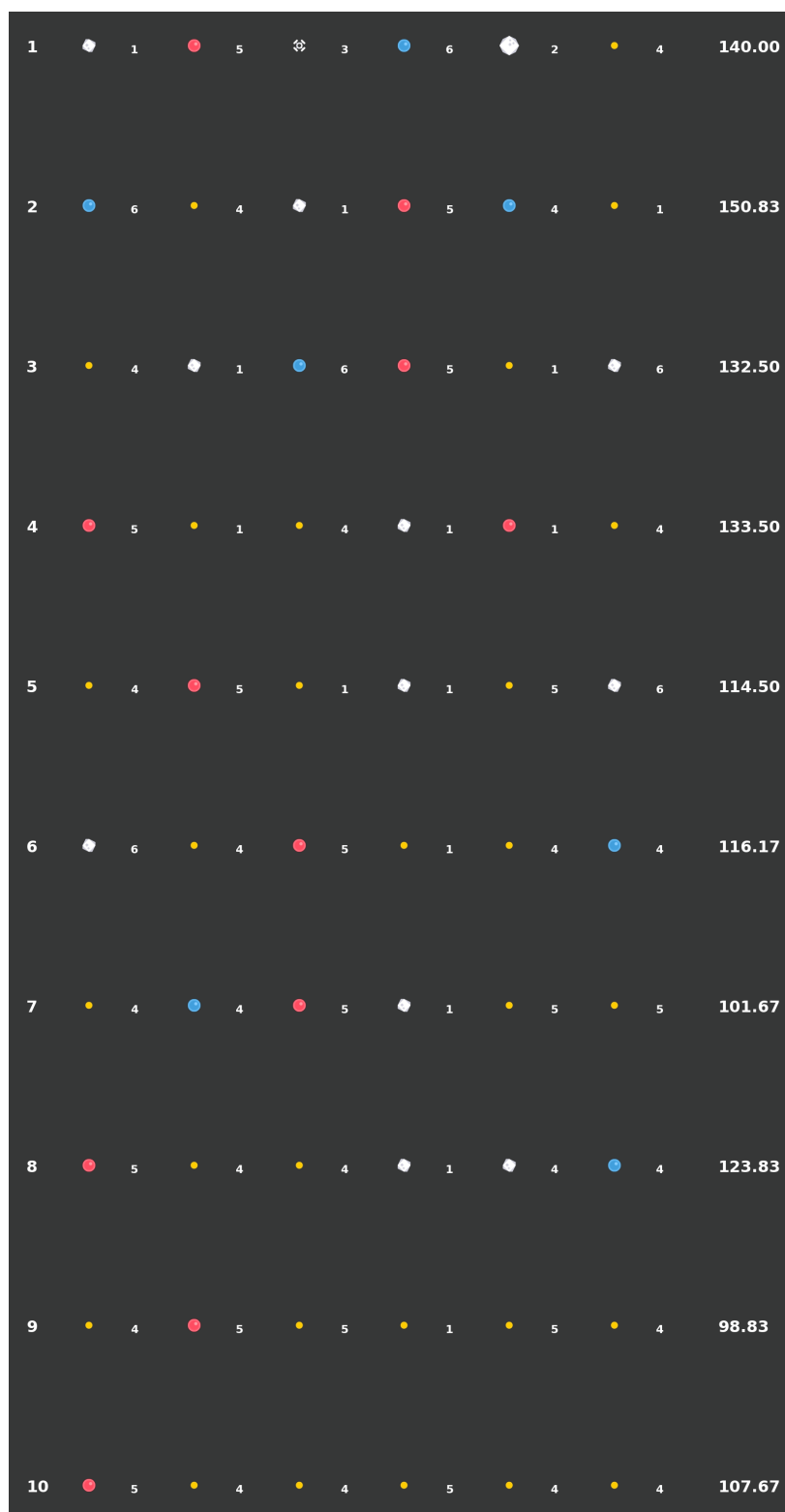


Figura E.4: Visualização da moda de cada onda com o fitness v3 contra Nave Movendo, Disparo Amarelo.

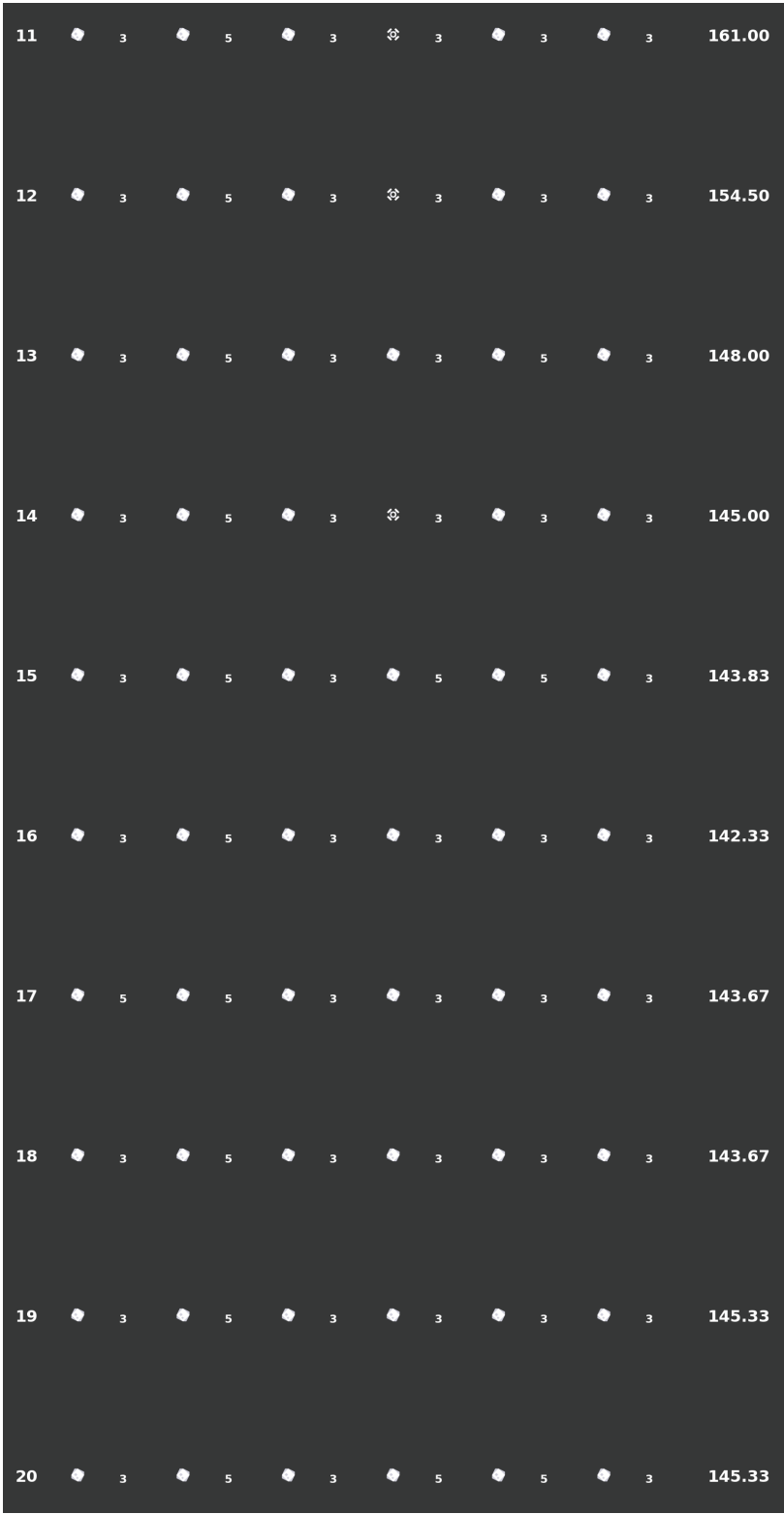


Figura E.5: Visualização da moda de cada onda com o fitness v3 contra Nave Movendo, Disparo Amarelo.

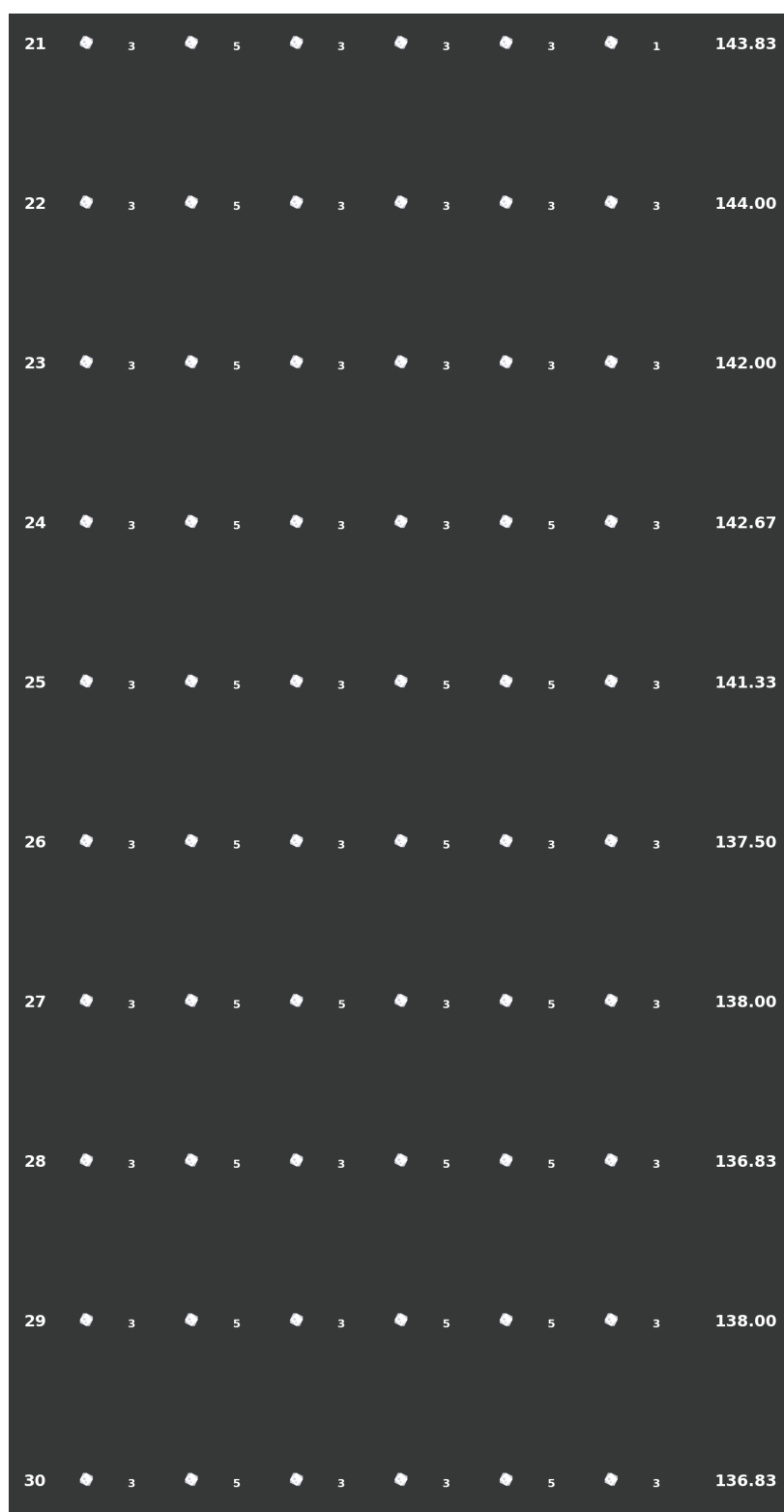


Figura E.6: Visualização da moda de cada onda com o fitness v3 contra Nave Movendo, Disparo Amarelo.

E.3 Nave Parada com Disparo Vermelho

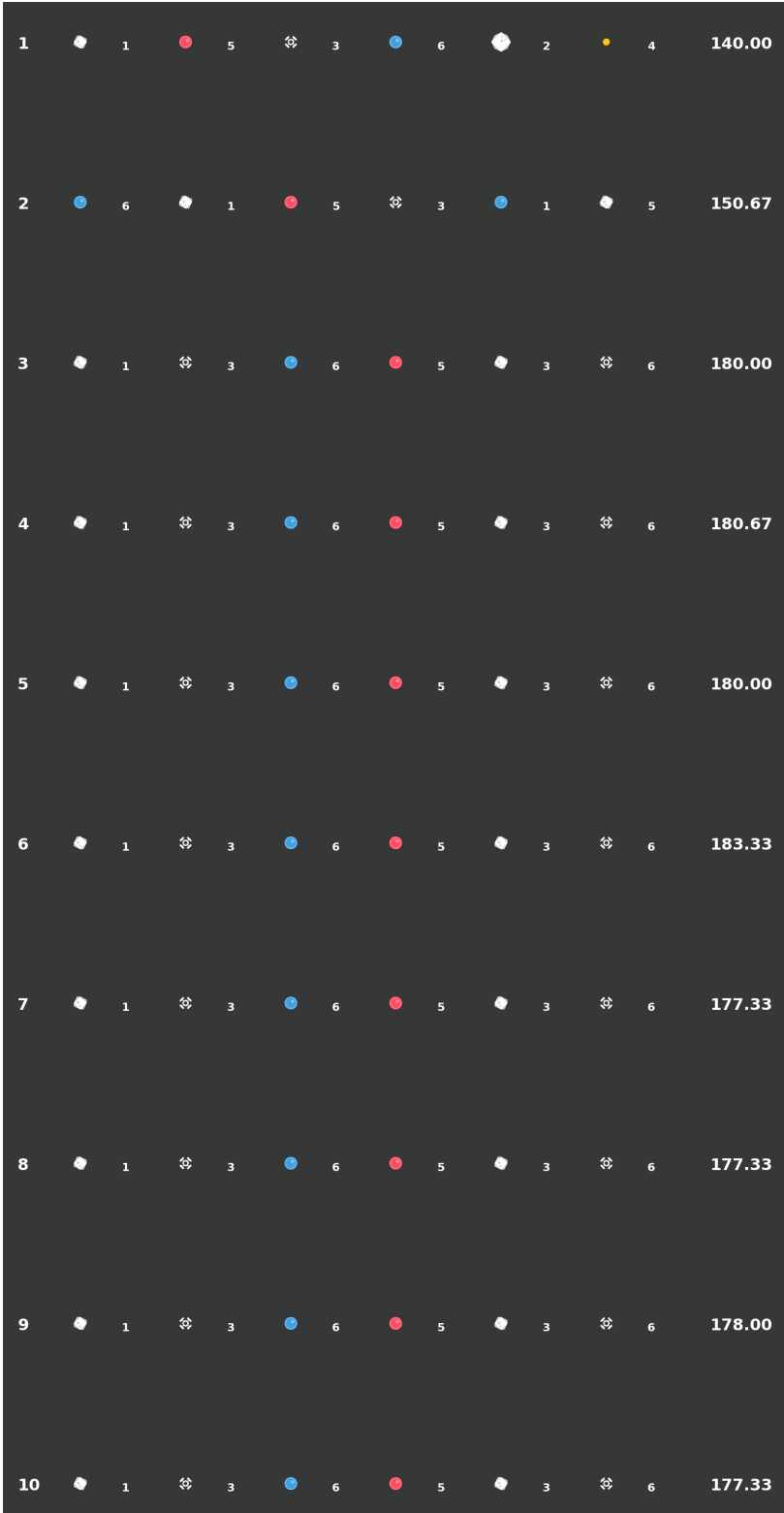


Figura E.7: Visualização da moda de cada onda com o fitness v3 contra Nave Parada, Disparo Vermelho.

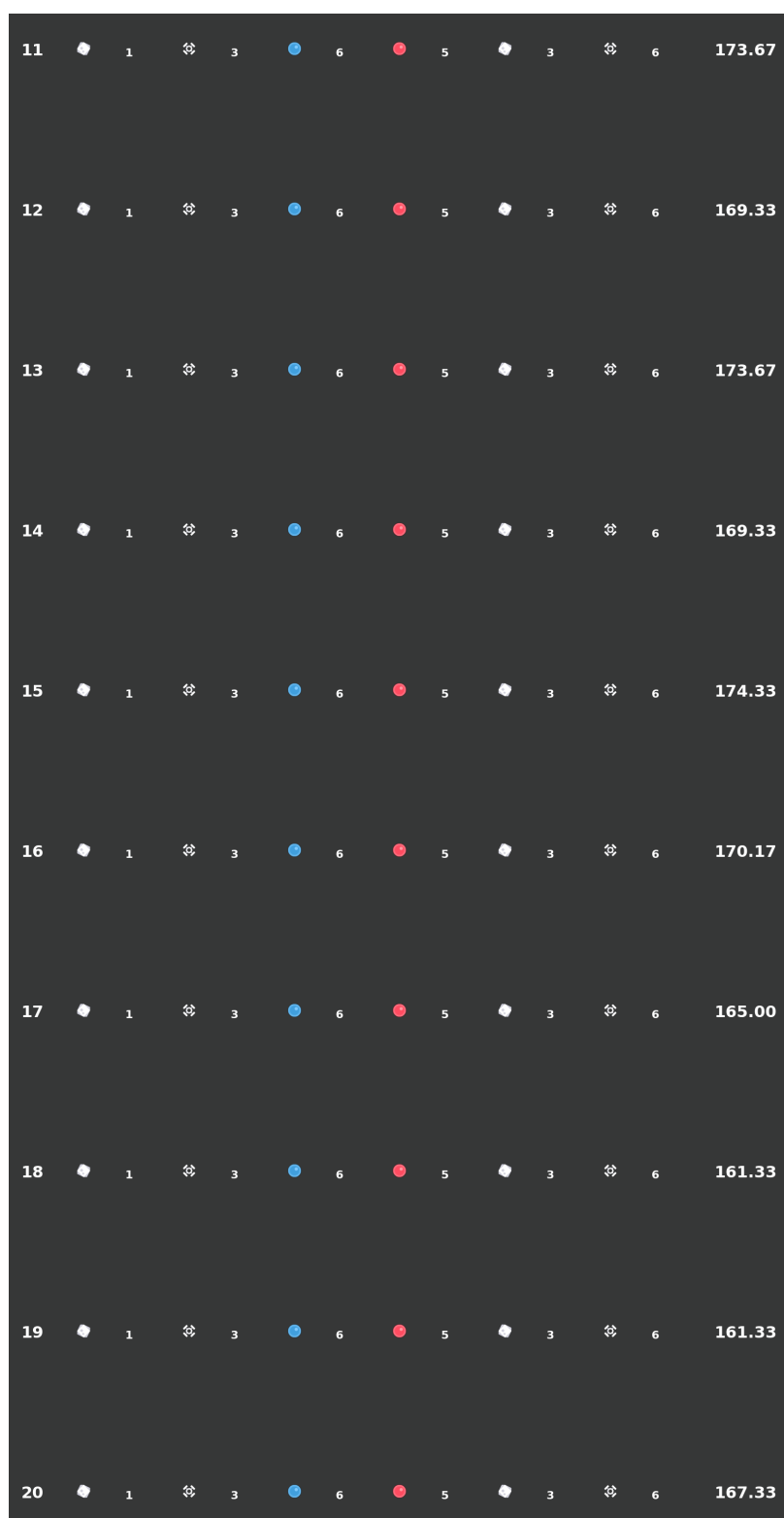


Figura E.8: Visualização da moda de cada onda com o fitness v3 contra Nave Parada, Disparo Vermelho.

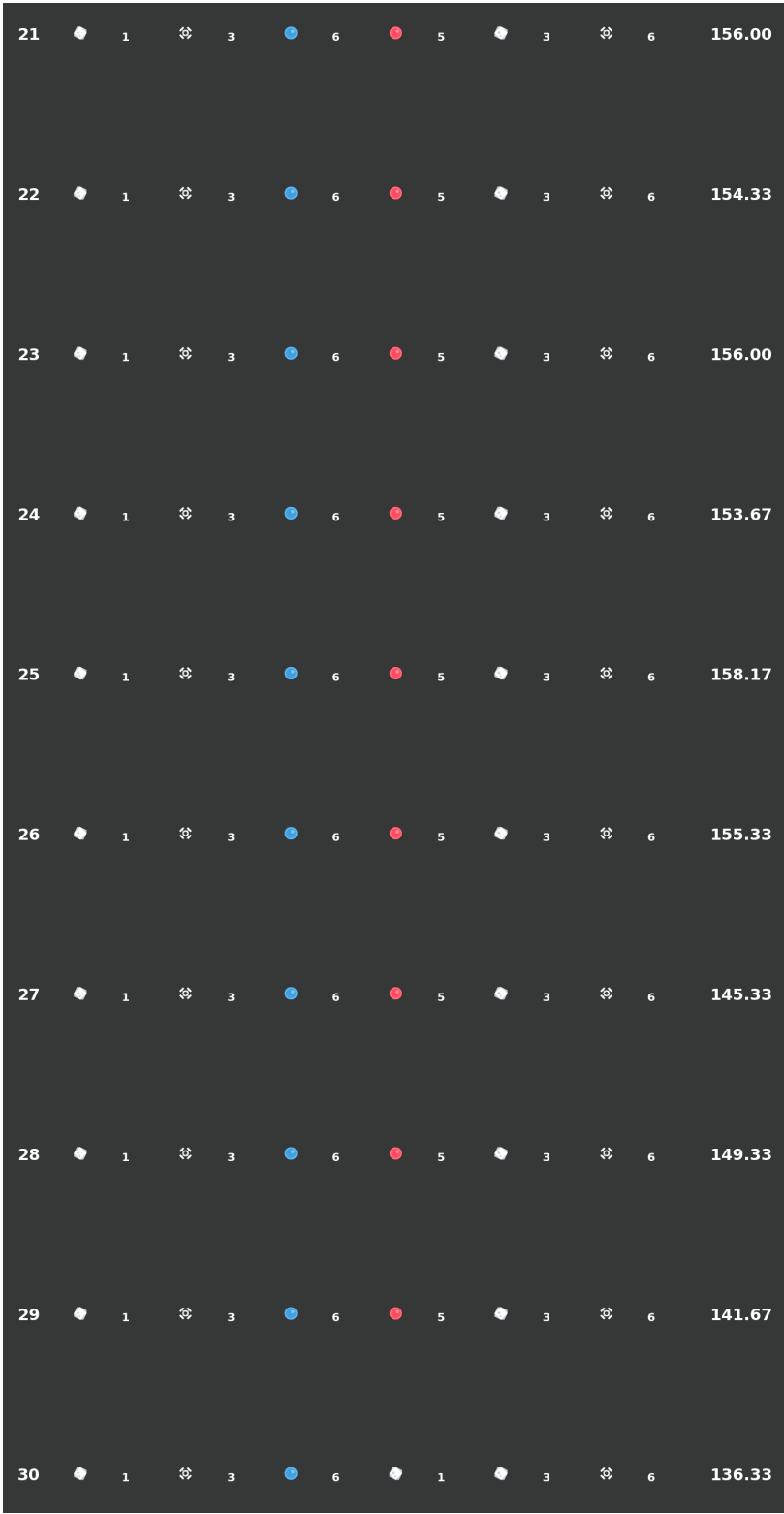


Figura E.9: Visualização da moda de cada onda com o fitness v3 contra Nave Parada, Disparo Vermelho.

E.4 Nave Movendo com Disparo Vermelho

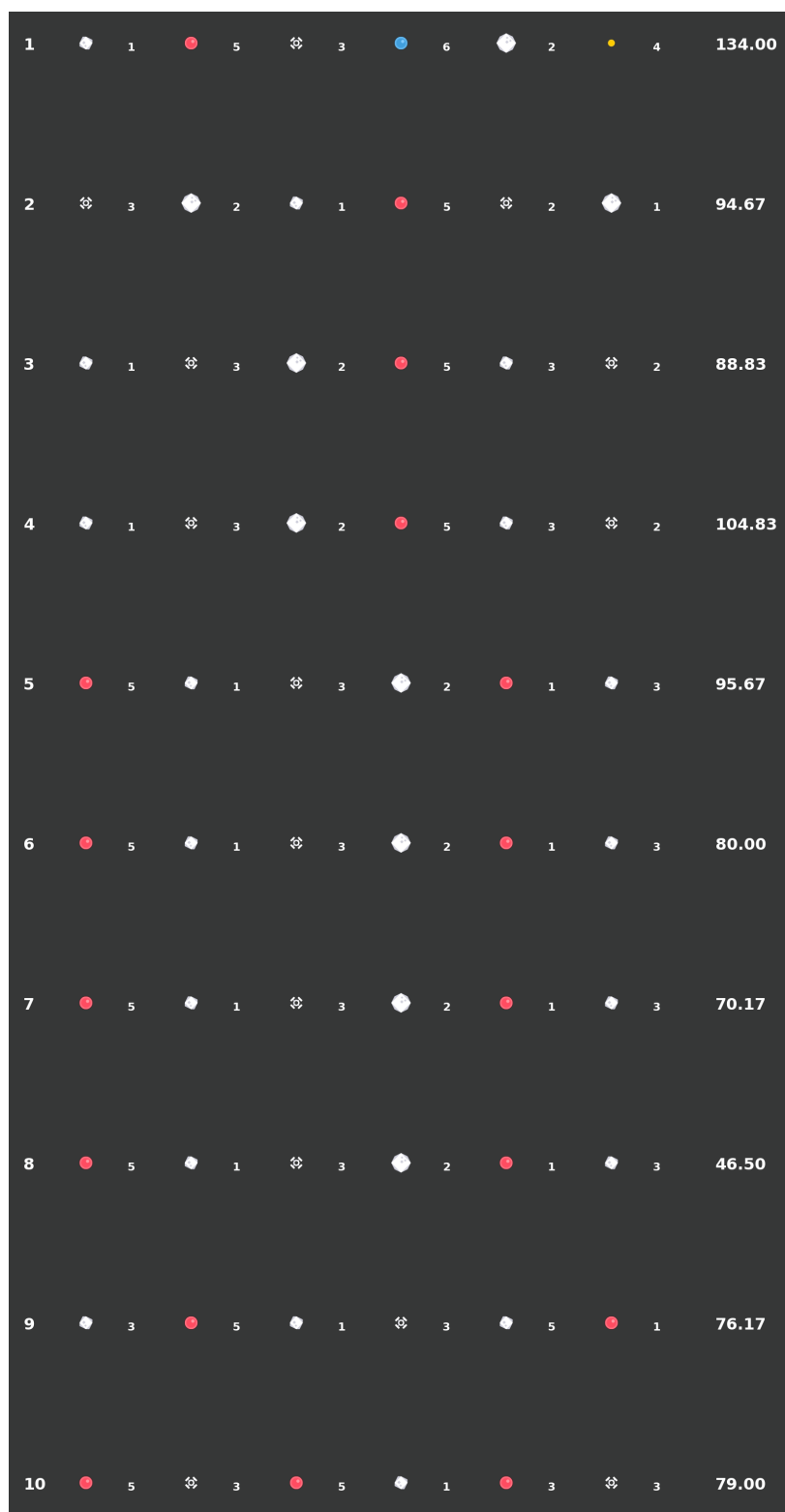


Figura E.10: Visualização da moda de cada onda com o fitness v3 contra Nave Movendo, Disparo Vermelho.

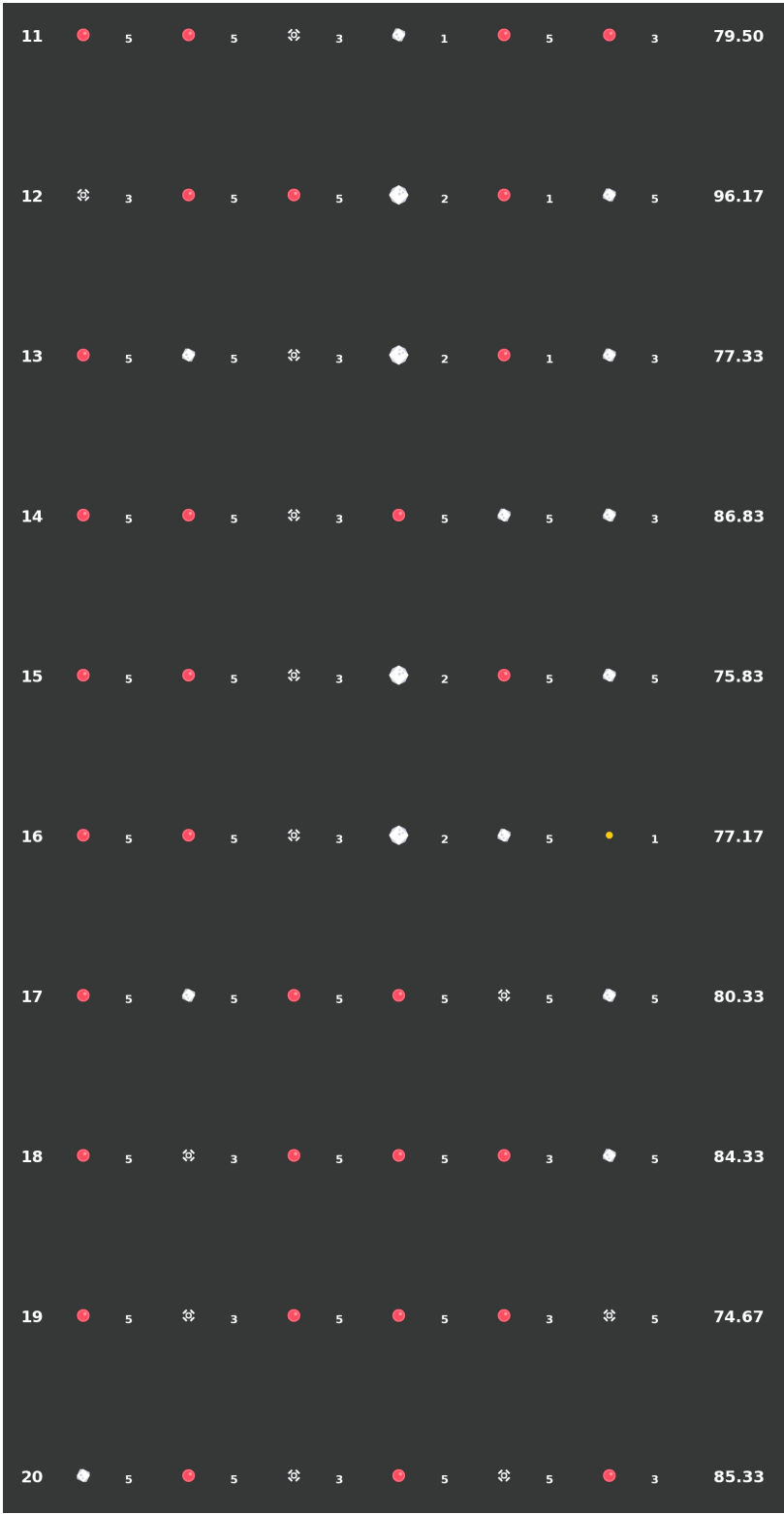


Figura E.11: Visualização da moda de cada onda com o fitness v3 contra Nave Movendo, Disparo Vermelho.

21		5		5		5		5		5		5	81.67
22		5		5		5		5		5		1	94.83
23		5		5		5		5		5		5	79.33
24		3		5		5		5		5		1	98.33
25		5		5		5		5		1		5	88.50
26		5		5		5		5		5		5	93.83
27		5		5		5		5		5		5	97.33
28		5		5		1		1		5		5	93.67
29		5		4		5		1		5		5	86.17
30		5		1		5		1		1		5	86.33

Figura E.12: Visualização da moda de cada onda com o fitness v3 contra Nave Movendo, Disparo Vermelho.

Referências

- [ANGELA e DANIEL 1999] D. ANGELA e V. DANIEL. *Design and analysis of experiments*. 2ª ed. Springer-Verlag, 1999 (citado nas pgs. 27, 28).
- [BALDWIN 2021] Mark BALDWIN. *Career Paths in the Game Industry*. URL: https://www.gamasutra.com/view/feature/2755/career_paths_in_the_game_industry.php?print=1 (acesso em 16/12/2021) (citado na pg. 1).
- [CAMPBELL e STANLEY 1963] D. T. CAMPBELL e J. C. STANLEY. “Experimental and quasiexperimental designs for research”. Em: (1963), pgs. 1–71. DOI: [10.1093/obo/9780195389678-0053](https://doi.org/10.1093/obo/9780195389678-0053) (citado na pg. 28).
- [CENTER 2021] Game Development CENTER. *Make a Tower Defense Game in Godot*. Ago. de 2021. URL: https://www.youtube.com/c/GameDevelopmentCenter/videos?view=0&sort=dd&shelf_id=0 (acesso em) (citado na pg. 11).
- [DEREK e ANDY 2008] Yu DEREK e Hull ANDY. *Spelunky*. 2008 (citado na pg. 1).
- [A. EIBEN e SCHIPPERS 1998] A. EIBEN e C. SCHIPPERS. “On evolutionary exploration and exploitation”. Em: *Fundam. Inform.* 35 (ago. de 1998), pgs. 35–50. DOI: [10.3233/FI-1998-35123403](https://doi.org/10.3233/FI-1998-35123403) (citado nas pgs. 24, 28).
- [A.E. EIBEN e SMITH 2015] A.E. EIBEN e J.E. SMITH. *Introduction to Evolutionary Computing*. 2ª ed. Springer, 2015 (citado na pg. 21).
- [GAD 2018] Ahmed GAD. *Genetic Algorithm Implementation in Python*. 2018. URL: <https://towardsdatascience.com/genetic-algorithm-implementation-in-python-5ab67bb124a6> (acesso em 16/12/2021) (citado na pg. 21).
- [HAUPT 2000] R.L. HAUPT. “Optimum population size and mutation rate for a simple real genetic algorithm that optimizes array factors”. Em: *IEEE Antennas and Propagation Society International Symposium. Transmitting Waves of Progress to the Next Millennium. 2000 Digest. Held in conjunction with: USNC/URSI National Radio Science Meeting (C. Vol. 2. 2000, 1034–1037 vol.2*. DOI: [10.1109/APS.2000.875398](https://doi.org/10.1109/APS.2000.875398) (citado nas pgs. 23, 24, 28, 46).

- [JANSEN *et al.* 2005] Thomas JANSEN, Kenneth A. De JONG e Ingo WEGENER. “On the choice of the offspring population size in evolutionary algorithms”. Em: *Evolutionary Computation* 13.4 (2005), pgs. 413–440. ISSN: 1063-6560. DOI: [10.1162/106365605774666921](https://doi.org/10.1162/106365605774666921) (citado na pg. 23).
- [JENSEN 2020] K. Thor JENSEN. *How Spelunky Made Procedural Generation Fun*. Set. de 2020. URL: <https://www.pcmag.com/news/how-spelunky-made-procedural-generation-fun> (acesso em 16/12/2021) (citado na pg. 2).
- [KENNEY 2021] KENNEY. *kenney*. 2021. URL: <https://www.kenney.nl/> (acesso em 20/12/2021) (citado na pg. 11).
- [KENT 2001] S. L. KENT. *The ultimate history of video games: from Pong to Pokemon – the story behind the craze touched our lives and changed de world*. 1ª ed. Three Rivers Press, 2001 (citado na pg. 1).
- [LINIETSKY e MANZUR 2018a] Juan LINIETSKY e Ariel MANZUR. *Frequently asked questions*. 2018. URL: <https://docs.godotengine.org/en/2.1/about/faq.html> (acesso em 01/12/2021) (citado na pg. 10).
- [LINIETSKY e MANZUR 2018b] Juan LINIETSKY e Ariel MANZUR. *Your first 2D game*. 2018. URL: https://docs.godotengine.org/en/latest/getting_started/first_2d_game/index.htm (acesso em 01/12/2021) (citado na pg. 11).
- [LYLE e ED 1979] Rains LYLE e Logg ED. *Asteroids*. 1979 (citado na pg. 5).
- [MAGALHAES e LIMA 2010] MARCOS NASCIMENTO MAGALHAES e ANTONIO CARLOS PEDROSO DE LIMA. *Noções de Probabilidade e Estatística*. 7ª ed. EDUSP, 2010 (citado na pg. 27).
- [MALLAWAARACHCHI 2017] Vijini MALLAWAARACHCHI. *Introduction to Genetic Algorithms – Including Example Code*. Jul. de 2017. URL: <https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3> (acesso em 16/12/2021) (citado na pg. 21).
- [NISHIKADO 1978] Tomohiro NISHIKADO. *Space Invaders*. 1978 (citado na pg. 5).
- [REIS e CAVICHIOLO 2014] L. J. de A. REIS e F. R. CAVICHIOLO. “Dos single aos multiplayer: a história dos jogos digitais”. Em: *LICERE - Revista Do Programa De Pós-graduação Interdisciplinar Em Estudos Do Lazer* 17.2 (jun. de 2014), pgs. 312–350. DOI: doi.org/10.35699/1981-3171.2014.858 (citado na pg. 1).
- [SHAKER *et al.* 2016] Noor SHAKER, Julian OGELIUS e Mark J. NELSON. *Procedural Content Generation in Games*. 1ª ed. Springer International Publishing Switzerland, 2016 (citado na pg. 1).
- [STEVE 1962] Russell STEVE. *Spacewar*. 1962 (citado na pg. 5).
- [STUDIO 2011] Ironhide Game STUDIO. *kingdom-rush*. 2011 (citado na pg. 4).

REFERÊNCIAS

- [TsANG 2021] Denise TsANG. “Innovation in the british video game industry since 1978”. Em: *Business History Review* 95.3 (2021), pgs. 543–567. DOI: [10.1017 / S0007680521000398](https://doi.org/10.1017/S0007680521000398) (citado na pg. [1](#)).