Pontifícia Universidade Católica
do Rio de Janeiro

Resolvendo Flappy Bird utilizando Algoritmo Genético com Rede Neural

Matheus Lourenço Fernandes Soares

PROJETO FINAL DE GRADUAÇÃO
Orientação Prof. Jônatas Wehrmann

CENTRO TÉCNICO CIENTÍFICO - CTC
DEPARTAMENTO DE INFORMÁTICA

Curso de Graduação em Ciência da Computação

Rio de Janeiro, junho de 2021.

Agradecimento

Primeiramente preciso agradecer à minha mãe, Nathalie, que me deu todo o suporte durante a minha vida, mesmo que em muitos momentos eu tenha acrescentado alguns fios de cabelo branco em sua cabeça. Sei o quanto batalhou para que eu tivesse a melhor formação possível e por isso serei eternamente grato, afinal estou me formando na PUC-Rio e no curso que amo. Além disso, preciso agradecer, também, por junto à Iaiá criar a pessoa que sou hoje. Aliás, Iaiá, obrigada por estar comigo desde o meu primeiro dia de vida e ser a representação de avó para mim.

Ao meu pai, por me ensinar todo o amor ao Flamengo que precisei ao longo do curso para seguir a diante e por me incentivar a sempre ser ético acima de tudo. À minha avó, Ilda por todos os almoços de domingo com a melhor costela do Brasil e de Portugal. Ao meu avô, Clóvis, que deixou saudades mas tenho certeza que celebraria essa conquista comigo com aquele abraço forte. Mas principalmente, agradeço aos dois como um casal que me mostrou o que é companheirismo dentro de uma relação e que um casamento só vai longe se fizerem piadas sobre o outro todos os dias. E tia Márcia, obrigada pelo melhor bolo de chocolate do mundo e por sempre me receber com um sorriso largo!

À Kel por ser a minha primeira irmã e dividir tanto da vida comigo. São inúmeros bons momentos que compartilhamos e iremos compartilhar juntos. Ao Carlos por participar tanto da minha criação e junto da minha mãe me dar o presente - e furacão - que é a minha irmã, Ana Beatriz.

Aos meus amigos, JG, Gabriel, Raphael, Minka, Fernanda, Júlia e Doia que estão comigo desde a quinta série e sabem bem como essa conquista foi esperada. Ao Bendia, Pedro e Gabrielzinho que foram os melhores amigos de faculdade que eu poderia receber nessa jornada. E todos os outros que em breve estarão comigo celebrando não só essa formatura, mas também o meu casamento.

Ao meu orientador, Jônatas Wehrmann, por me aceitar como orientando, por me apoiar nesse projeto e por auxiliar em todo o processo de escrita. Muito obrigado mesmo! E a todos os professores da PUC-Rio pela formação excelente que tive nesse local e a todos os profissionais com quem já trabalhei por ajudarem a construir o profissional que sou hoje.

Por último, à May, minha noiva. Te amo mais que tudo. Muito obrigado por estar nesses últimos oito anos comigo e me apoiar em todos os momentos que precisei. O resto desse agradecimento, você precisará esperar pelos nossos votos para saber. Mas posso adiantar: mal posso esperar para ter todo o resto da minha vida ao seu lado.

Resumo

Soares, Matheus L. F. Wehrmann, Jônatas. **Resolvendo Flappy Bird utilizando Algoritmo Genético com Rede Neural**. Rio de Janeiro, 2021. 18p. Relatório de Projeto Final II - Departamento de Informática. Pontifícia Universidade Católica do Rio de Janeiro.

O presente projeto descreve o processo de desnvolvimento do aplicativo Flappy AI desenvolvido para plataforma iOS. No aplicativo é possível jogar o jogo Flappy Bird e também assistir como um Algoritmo Genético com Rede Neural consegue evoluir para resolver o problema proposto pelo jogo. O aplicativo foi desenvolvido completamente usando Swift e a biblioteca de jogos 2D da Apple chamada SpriteKit.

Palavras-chave: iOS, Swift, SpriteKit, Algoritmo Genético, Rede Neural, Flappy Bird

Abstract

Soares, Matheus L. F. Wehrmann, Jônatas. **Solving Flappy Bird with Genetic Algorithm and Neural Network**. Rio de Janeiro, 2021. 18p. Relatório de Projeto Final II - Departamento de Informática. Pontifícia Universidade Católica do Rio de Janeiro.

The Project describes the development process of creating Flappy AI developed for iOS. In this App it is possible to play the Flappy Bird game and also watch how a Genetic Algorithm with Neural Networks manages to evolve to solve the proposed problem. The App was developed completely using Swift and an Apple framework to make 2D games called SpriteKit.

Key-words: iOS, Swift, SpriteKit, Genetic Algorithm, Neural Networks, Flappy Bird

Sumário

1 Introdução

A área de Inteligência Artificial é muito bem definida por Stuart Russel e Peter Norvig em seu livro Inteligência Artificial como "o estudo de agentes que percebem seu ambiente e conseguem realizar ações" [1] . Sendo assim, Inteligência Artificial é a área da computação focada na construção de algoritmos que são capazes de, dada uma série de opções, analisar o ambiente em que estão inseridos e tomar uma decisão que, se não a melhor, a que este agente consideraria a melhor.

Uma maneira efetiva e interessante de se testar um algoritmo ou uma teoria para uma Inteligência Artificial (IA) é através da criação de jogos eletrônicos, pois neles temos um ambiente favorável, dinâmico e competitivo que permite o aprendizado e os testes de escolhas. [2] No início, a maior parte dos trabalhos envolvendo jogos e IAs era voltado para os jogos de tabuleiro, sendo um grande exemplo disso o algoritmo da IBM chamado de *Deep Blue*, que protagonizou um marco das interações de IAs em xadrez quando em 1997 ganhou do campeão mundial Garry Kasparov em uma partida exibicional. Hoje, existe o Alpha Zero [3] , desenvolvido pela empresa Deep Mind, que é considerado o melhor algoritmo capaz de jogar xadrez do mundo. Ele foi criado usando a técnica de *deep reinforcement learning*, e conseguiu jogar 44 milhões de jogos contra si mesmo nas primeiras 9 horas de aprendizado.

O uso de jogos eletrônicos para testar machine learning vem, então se mostrando eficaz e mais usado a cada dia, com jogos conhecidos que passam desde Pacman, Super Mario até *Counter-Strike*. [4] O pioneiro do uso de redes neurais em IA para jogos eletrônicos foi o algoritmo da Google, o DeepMind, que através de deep reinforcement learning foi capaz de jogar os jogos de *Atari 2600* e conseguir obter mais pontos que qualquer recordista humano em três dos sete jogos disponíveis no *Atari 2600*. [5]

O objeto do presente trabalho, o jogo *Flappy Bird*, por exemplo, está presente em outras centenas de artigos que visam estudar, conhecer e analisar a criação de uma Inteligência Artificial. *Flappy Bird* é um jogo mobile criado pelo desenvolvedor vietnamita Dong Nguyen que fez muito sucesso em 2013, sendo um dos jogos mais baixados na história. Consiste em um pássaro de animação 2D que o jogador precisa manter vivo pelo maior tempo possível através de cliques na tela para equilibrar o peso do pássaro e passá-lo por obstáculos, que são canos similares aos de Super Mario; justamente pela simplicidade de suas regras foi o jogo escolhido para esse trabalho. [6] No presente projeto, será usado deep reinforcement learning e algoritmo genético para criar um agente capaz de jogar e se manter vivo pontuando o maior recorde de tempo possível no jogo.

Com isso, o presente projeto tem como objetivo final o desenvolvimento de um agente que utiliza Inteligência Artificial capaz de jogar o jogo Flappy Bird, que desenvolvi desde o início com todas as mecanicas adaptadas do jogo original. Visando o uso em iOS, utilizei da linguagem Swift para criação de três diferentes modos de jogo: Player mode, AI mode e Versus mode. Ao fim do projeto se espera que o agente desenvolvido seja capaz de bater o recorde de um jogador humano que seja expert no jogo.

2 Flappy Bird Original

Flappy Bird foi lançado em maio de 2013 pela companhia de desenvolvimento de jogos dotGears e consiste em um pássaro voando na horizontal a uma velocidade constante, contudo esse pássaro cai e oscila na altura, como se fosse puxado pela gravidade. Quem controla a altura de voo é o jogador, através de cliques na tela. O objetivo do jogo é passar com esse pássaro através de espaços na vertical formados por dois canos. Esses espaços verticais variam, mas a distância entre os canos na horizontal não. É dado fim de jogo quando o pássaro atinge o chão ou encosta em um dos canos.

O jogo é simples, com regras e dinâmica fáceis, contudo, é muito comum que mesmo após horas de jogo, um jogador não consiga fazer mais que 10 pontos. O que fez com o jogo se tornasse viciante, chegando a um patamar de problema para muitas pessoas, e por isso, o seu criador acabou retirando do ar apenas oito meses após o seu lançamento, em janeiro de 2014.

Desde então, o jogo vem sendo recriado por outros desenvolvedores e virou objeto de estudo para machine learning, bem como também para cálculos afim de melhorar a experiência do usuário, facilitando um pouco a dinâmica do jogo. [7] Por esses motivos e pela vasta literatura acerca do assunto, acabou se tornando objeto desse projeto.

3 IA em jogos

Em 1997 um artigo foi publicado abordando a criação de uma IA feita para jogar Gamão e que aprendeu apenas jogando contra si mesma e olhando seus próprios resultados. Essa IA desenvolvida com base no algoritmo de reinforcement learning alcançou pontuações maiores que os jogadores humanos mais bem conceituados no Gamão. [8]

Contudo, apesar dos casos mais famosos de IA em jogos serem com Damas, Xadrez, ou o próprio Gamão, esses jogos vem de uma fórmula básica: avaliação linear de muitos recursos simples, treinados pela repetição de aprendizados com diferença temporal e combinados com um algoritmo de busca adequado.

Mas foi David Silver que pela primeira vez trouxe a estratégia para um jogo com um agente de Inteligência Artificial, no jogo chinês de mais de 2 mil anos, GO. Usando redes neurais profundas treinadas por uma combinação de aprendizado com dados de jogos de experts e com resultados de jogos contra si mesma, a IA conseguiu pela primeira vez na história derrotar um jogador humano profissional no jogo de tamanho normal de Go. [9]

Além dos já citados, temos dois exemplos bem famosos de uso de Inteligência Artificial em jogos eletrônicos: o da empresa OpenAI a qual desenvolveu uma IA para um jogo altamente complexo chamado Dota 2. O projeto chamado OpenAI Five iniciou vencendo partidas um contra um e hoje já é capaz de vencer sozinho no modo principal, onde se joga cinco contra cinco, contra times campões mundiais [10] . Outro exemplo foi o desenvolvimento de um agente, criado pela empresa responsável pelo projeto do Alpha Zero no xadrez, a Deep Mind, capaz de jogar o jogo StarCraft II da blizzard. Este projeto obteve um ranking entre os melhores 99.8% dos jogadores de acordo com o ranking do próprio jogo [11] . Em ambos os exemplos a abordagem foi o uso do algoritmo de Deep Reinforcement Learning que foi usado nesse projeto.

Seguindo o conceito usado pela IA que conseguiu se desempenhar muito bem no Atari 2600, Naveen Appiah and Sagar Vare elaboraram uma IA para o jogo Flappy Bird em 2016. Usando deep reinforcement learning, eles conseguiram uma IA que não se saiu muito bem na pontuação, [12] entretanto. Outro projeto que utilizou do jogo Flappy Bird e também de Support Vector Machine (SVM), conseguiu melhores resultados no tempo de sobrevivência do agente de IA, contudo, nesse projeto os dados de treinamento foram inseridos e produzidos de forma manual. [13]

No presente projeto, através do uso em conjunto de deep reinforcement learning e algoritmo genético, produzi um conteúdo diferente do encontrado na literatura, uma vez que faz uso de fantasmas do pássaro, cada um com a sua rede neural, tentanto ir mais longe no jogo e sobreviver por mais tempo. Além disso, produzi um modo em que a IA pode jogar contra o jogador e os dois tentam ser melhor que o seu oponente.

4 Algoritmos Referências

4.1 *Deep reinforcement learning*

Reinforcement learning funciona através da criação de diferentes agentes que interagem com o ambiente em que estão inseridos. Esse ambiente nos fornece informações sobre a capacidade e a taxa de sucesso das estratégias dos agentes, que acabam por experenciar diversas estratégias afim de ter um portfólio grande o suficiente para tomar decisões cada vez mais acertadas. O deep reinforcement learning é escolhido quando há um espaço muito grande para ser reconhecido.

Redes neurais são sistemas com vários nós capazes de transformar um ou vários inputs em um ou vários outputs. Através deles podemos mudar constantes que são usadas para tomar estas decisões sem mudar a arquitetura do projeto. Desta forma para algoritmos de aprendizado de máquina, estes sistemas são essenciais pois dão a habilidade de se adaptar ao problema a cada iteração e após diversas tentativas podem chegar numa solução esperada.

Em aprendizado de máquina temos 3 principais tipos de formas que o software é capaz de aprender e evoluir:

- Aprendizado supervisionado - É o tipo de aprendizado em que é dado uma série de inputs e os outputs esperado para treinar o sistema.
- Aprendizado não supervisionado - Assim como no aprendizado supervisionado, o sistema tenta encontrar padrões, contudo, sem os dados previamente fornecidos, podendo encontrar padrões não esperados através da exploração de dados.
- Aprendizado por reforço - Este tipo, foca em tentar conseguir o máximo de recompensas enquanto explora um ambiente fazendo ações.

Para o aprendizado reforçado alguns conceitos são importantes para que um agente possa tomar uma decisão:

- Ambiente - Para Flappy Bird são os espaços na vertical entre dois canos e a visão dos próximos canos que se aproximam na horizontal. Além da altura e gravidade do próprio pássaro.
- Ações - Tomar a decisão se irá pular ou não, obter sucesso ao atravessar o espaço entre os canos e até mesmo a ação de ir de encontro ao chão, perdendo o jogo.
- Recompensa - É o retorno se a ação tomada foi positiva ou negativa para o agente. Nesse caso, se obteve sucesso ao se manter vivo, passando pelos espaços entre os canos, não encostando em cano algum nem indo de encontro ao chão, são ações que merecem recompesa. Do contrário, gera o fim do jogo e não há recompensa.

Ambos os conceitos de redes neurais e aprendizado reforçado podem ser unidos para contruir um tipo de agente que é capaz de aprender com os erros e chegar no objetivo desejado através de um algoritmo que o recompensa por acertos. É com essa abordagem que projetos como o Alpha Zero [3] no xadrez e o OpenAI five [10] no Dota 2 foram capazes de vencer de campeões mundiais em seus respectivos jogos.

4.2 *Algoritmo genético*

O algoritmo genético funciona como um modelo matemático da Teoria da Evolução de Darwin, desenvolvido por John Henry Holland em 1975 e foi popularizado por David Goldberg 14 anos depois.[14] Esse algoritmo segue a premissa de aprendizado a partir de um sistema de soluções que tenta obter sucesso frente a um problema. Ou seja, os melhores vão sobreviver. Este método de solução de problemas propõe uma abordagem de tentativa e erro para garantir a resolução do problema.

Ao implementar este tipo de algoritmo é criado uma série de individuos que é denominado como uma geração. A população desta geração compete entre si para achar a melhor solução do problema proposto. Uma vez que todos os individuos chegaram a uma resolução do problema ou falharam para encontra-la, o algoritmo determina quais foram os individuos que chegaram mais próximo da solução através de uma função chamada Função de *fitness*.

Uma vez determinado os melhores individuos da geração, é nascida uma nova geração com individuos completamente novos usando técnicas de mutação e cruzamento, que são pertinentes ao algoritmo genético. O cruzamento é responsável por selecionar os melhores traços dos individuos da geração anterior e cruzar com outros traços de outros individuos. Já a mutação tem como objetivo garantir que a solução do problema não caia em uma solução ótima local e que a população converta apenas para esta solução, ajudando na procura da melhor solução possível para o problema em questão.[15]

Uma vez gerada a população, o algoritmo volta ao início e tenta fazer com que esta nova geração resolva o problema. A abordagem do algoritmo genético é capaz de resolver problemas difíceis sem necessariamente haver um treinamento prévio e se bem implementado, com o passar das gerações é possível ver a solução do problema avançando gradativamente.

5 Funcionalidades do sistema

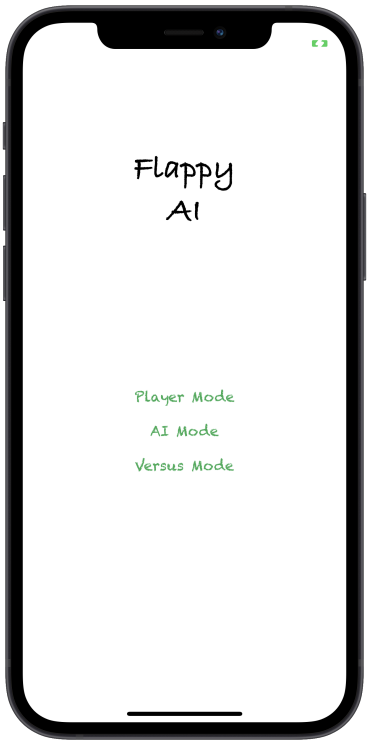

Figura 1: Menu principal

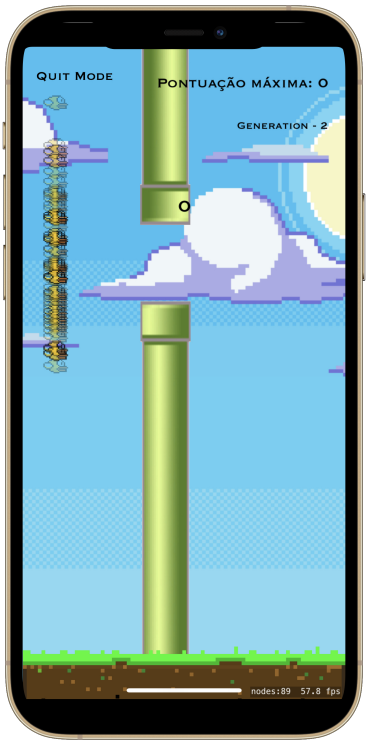

Figura 2: Jogo

O jogo possui uma primeira tela em cor básica que aprentas os três principais modos de jogo dispostos escritos por extenso no meio da tela, conforme Figura 1 acima. Cada modo tem a sua funcionalidade, mas segue as mesmas regras e o mesmo layout. São eles:

5.1 *Player Mode*

O primeiro modo do jogo é bem similar ao que foi lançado em 2013. Com adaptação de layout, o jogador tem a possibilidade de jogar um jogo normal de Flappy Bird, no qual a cada espaço entre os canos que ele conseguir ultrapassar sem encostar é contabilizado como um ponto para o jogador que é exibido no canto superior direito. Salvamos localmente a pontuação máxima do jogo para que o jogador tente bater o próprio recorde.

O app foi testado por pessoas diferentes e a maior pontuação realizada até o presente momento, em um período de dois meses de jogo, foi de 15 pontos.

Abaixo a Figura 3 representa o jogo nesse modo.

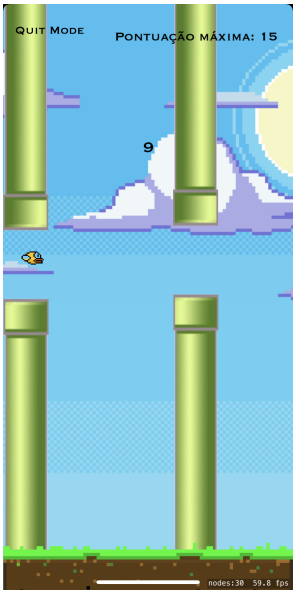

Figura 3: Player Mode

5.2 *AI Mode*

Neste modo o jogador não atua e podemos ver o algoritmo genético funcionando. O jogo começa com vários "fantasamas"aleatórios de pássaros. Cada fantasma possui a sua rede neural própria e única e tenta sobreviver ao jogo, cada um desses fantasmas joga escolhendo quando é a melhor hora de executar a ação do jogo: pular. Alguns pássaros vão melhor que outros e uma vez que todos os pássaros morrem uma nova geração de pássaros é gerada.

Os pássaros da nova geração são criados a partir dos melhores passáros da geração anterior, com algumas pequenas mudanças em suas redes neurais, ou seja mutações, para que possam tentar ser ainda melhores que seus antepassados.

Neste modo temos um botão de sair do modo no canto superior direito, logo abaixo temos o de vida dos pássaros a cada geração. Temos também um contador do lado superior direito que mostra a melhor run de antigas gerações mas não salva de outras runs. Por último, podemos ver qual a atual geração em uma label na parte superior esquerda da tela, conforme pode ser observado nas imagens abaixo.

Figura 4: Segunda geração

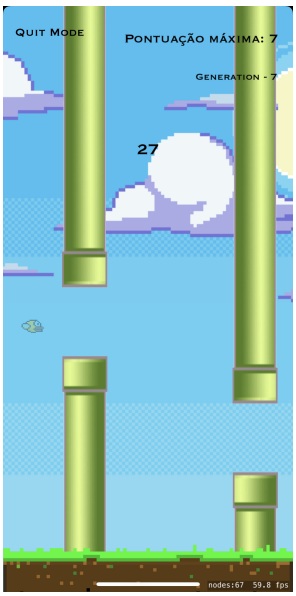

Figura 5: Sétima geração

5.3 *Versus Mode*

Este modo, que aparece somente após ter jogado o modo IA pelo menos uma vez, o jogador pode jogar como no Player Mode. Contudo, juntamente do seu pássaro aparece um fantasma do melhor pássaro que a IA conseguiu fazer enquanto estava no AI Mode. Assim, o jogador e a IA jogam como oponentes, tentando obter melhor pontuação que o seu adversário. Na figura 6 podemos ver que o jogador é pássaro opaco e o seu oponente, IA, é o pássaro translúcido.

Figura 6: Versus Mode

6 Especificação do sistema

6.1 *Tecnologias usadas*

O sistema foi desenvolvido para a plataforma iOS e a linguagem de programação escolhida para fazer o desenvolvimento foi Swift, criada pela Apple e com foco no desenvolvimento para plataformas da empresa. O ambiente de desenvolvimento foi um Macbook Pro 16' com 32 GB de memória RAM e processador Intel i9 de 2,3 GHz e foi testado em um iPhone 12.

Outra ferramenta importante no desenvolvimento foi a biblioteca do SpriteKit da Apple que auxilia a criação de jogos em 2D. Com o uso dessa ferramenta pude adicionar os elementos do jogo como pássaros, canos e o chão, adicionar velocidade ao cano que se movimenta em direção latitudinal ao pássaro, aplicar o efeito da gravidade somente para o pássaro, aplicar um impulso ao pássaro uma vez que o usuário clica na tela e também ajuda na detecção de contato entre os elementos do jogo, permitindo retornar se o jogador perdeu ou se permanece vivo.

6.2 *Arquitetura*

O sistema segue uma arquitetura bem comum para aplicativos mobile que é a arquitetura de *Model*, *View* e *Controller* (MVC). Nesta arquitetura temos o *Model*- que chamaremos de modelo- responsável por gerenciar os dados e estados das entidades presentes no jogo, o *Controller* é onde a lógica do jogo fica isolada e a *View* fica responsável pela apresentação do jogo.

A maioria dos modelos que temos no sistema herdam de SKSpriteNode, uma classe do SpriteKit que consegue ser renderizada na tela uma vez informada alguns dados como tamanho, textura e posição. Após a inicialização também é possível dar velocidade, se deve ou não ser afetada pela gravidade e uma representação de com quais outras entidades este modelo deve interagir caso detecte uma contato. A figura a seguir representa parte dos modelos do jogo:

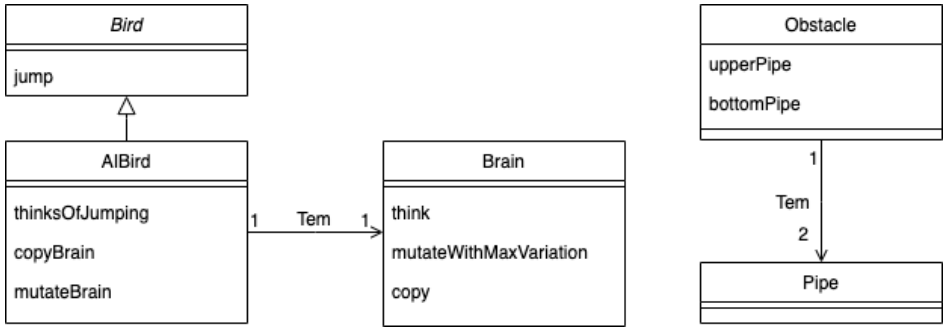

Figura 7: Model

- *Brain* - Este modelo é responsável por fazer a tomada de decisão de um *AIBird* se ele deve pular ou não usando o método think. Nele ficam salvos os pesos que serão usados na hora de decidir. Também temos algumas outras funções como a de mutateWithMaxVariation que muda os valores dos pesos aleatoriamente levando em conta uma variação máxima.
- *Bird* - É o pássaro usado no modo Player Mode, este modelo herda de SKSpriteNode e possui algumas caracteristicas como ser afetado pela gravidade e um método de pular que dá um impulso vertical para cima no pássaro
- *AIBird* - Este modelo herda de pássaro e possui as mesmas características dele com o adicional de ter uma instância de Brain, permitindo a ele ter uma função que pensa se deve pular ou não dado um momento. Assim como para brain, temos uma função de mutar que chama a função de mutação de Brain.
- *Pipe* - Outro importante elemento do jogo, o modelo dos canos, assim como o de *Bird*, herda de SKSpriteNode porém não é afetado pela gravidade do jogo e na inicialização é dada uma velocidade horizontal.
- *Obstacle* - Um conjunto de dois *Pipes* forma um *Obstacle*, ou seja, um obstáculo a ser vencido, e nele tem salvo os dados do tamanho do espaço entre os *Pipes* e qual é a posição máxima e mínima na hora de gerar um novo obstáculo, a fim de que nenhum espaço entre os canos apareça fora da tela do celular.

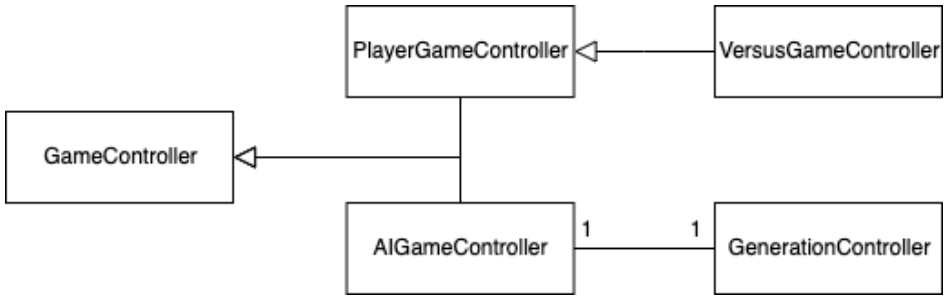

Figura 8: Controller

- GameController - Este Controller é responsável por controlar as mecânicas presentes em todos os modos de jogo. Ele que faz o controle de configuração inicial do jogo, configurando o botão de sair, background, geração de canos e chão do jogo.
- PlayerGameController - Aqui controlamos o que é específico para o modo de jogo Player Mode. Criando uma instância de Bird, configurando o pássaro para pular quando o usuário toca na tela e configurando o contador de pontuação.
- VersusGameController - Neste Controller, que herda de PlayerGameController, a única diferença para o PlayerGameController é que temos um *AIBird* além do Bird comum, e devemos inicializar ele com os pesos que salvamos quando rodamos um jogo no modo AI Mode.

- GenerationController - Este Controller possui toda a lógica de criação de gerações do modo AI Mode, além de controlar qual pássaro foi o melhor da geração para que a próxima tenha ele como descendente.
- AIGameController - Aqui temos um Controller que herda de PlayerGameController e que fica responsável por, com ajuda do GenerationController, configurar o jogo para o AI Mode. Ele deve iniciar um timer que a cada cem milissegundos chama um método que faz os pássaros calcularem se devem pular ou não, baseado em alguns inputs passados por este Controller.

Por último temos duas Views que são responsáveis por cada tela do jogo.

- MenuViewController - Nela temos o controle de três botões cada uma chamando a próxima tela e iniciando o jogo como o modo selecionado pelo usuário.
- GameViewController - Nesta View chamamos o controller baseado no modo selecionado na tela anterior e configuramos o controller para ser chamado quando o usuário toca a tela.

7 Implementação

7.1 *O jogo*

Como já foi falado anteriormente, o jogo base foi construído usando a biblioteca para contrução de jogos 2D da Apple chamada SpriteKit. Esta biblioteca usa o conceito de cena como a área de jogo, e é nesta cena que se define atributos do mundo do jogo como gravidade, tamanho da cena em comparação a tela e o que acontece uma vez que dois objetos da cena se colidem. Outro conceito importante para a biblioteca é o nó, que são os objetos do jogo, como o pássaro e o cano.

A maioria dos modelos do jogo herdam de SKSpriteNode, os nós da biblioteca, são eles:

- Pássaro, Cano e Chão - São nós que aparecem na tela que possuem caracteristicas diferentes entre si;
- Nó de pontuação - é um nó invisível presente entre dois canos que serve para que quando um pássaro passe por ele o jogo pontue um ponto;
- Teto - é um nó presente acima da tela presente para que pássaro não possam voar por cima dos canos;
- Nó de fim da tela - Nó presente na parte esquerda da tela para que uma vez que um cano saia da tela, ele possa ser retirado com o propósito de não consumir grande parte da memória do aparelho de forma desnecessária e assim, melhorar a performance do jogo;

Na inicialização de um nó, podemos configurar atributos como, por exemplo, se o objeto é dinâmico (caso do cano e pássaro) ou estático, se ele é afetado pela gravidade (caso apenas do pássaro), o tamanho e posição em que ele começa na cena. Além disso, um dos atributos mais importante a se configurar é o de *PhysicsBody*, onde é possível dar um "corpo"ao nó em que, com ele, a biblioteca é capaz de detectar contato entre nós.

A configuração de contato é feita usando uma estrutura de dados de *Bitmap*, onde deve-se definir para o nó o bit que o representa, configurar qual o *Bitmap* de colisão (para quando o jogo tem contato que muda a trajetória do objeto) e o *Bitmap* para contato. Para configurar corretamente foi feito para cada nó uma operação de Bit OR nos nós que se deseja ter alguma ação caso entrassem em contato. No caso do pássaro, como era necessário parar o jogo caso ele tivesse contato com o cano, chão ou teto, foi feito esta operação usando os bits configurados para estes três nós, enquanto que para o cano, a operação foi feita com o pássaro e com o nó de fim de tela.

Outro recurso importante da biblioteca que foi usado é o *SKAction* onde é possível criar uma ação ou sequencia de ações e executá-la quando necessário. Para a criação dos obstáculos, esta ferramenta foi usada criando uma sequência de ações onde primeiro se cria um obstáculo e em seguida espera 2.3 segundos. Após a criação desta sequência de ações, a biblioteca oferece a opção de realizá-la para sempre. Uma vez que a IA ficasse boa no jogo, tornou-se necessário a remoção de nós não usados, para isso foi criado o nó de fim de tela - que tem a finalidade de remover qualquer cano que em contato ao ultrapassar os limites de tela, com o propósito de manter o número de nós baixo.

Na implementação do *AI Mode*, algo que foi essencial entender é quantos nós é possível adicionar sem perder performance, visto que quanto maior fosse o número de pássaros por geração mais rápido o algoritmo

genético poderia encontrar a melhor solução possível. A biblioteca do *SpriteKit* tem uma ferramenta que indica quantos nós existem na cena e quantos Frames por segundo (FPS) a mesma está rodando. Usando tal da biblioteca *SpriteKit*, foi observado que o jogo no modo *Player Mode* roda em uma taxa entre 55 e 59 FPS e para que ele ficasse agradável de se ver, foi determinado que a menor taxa aceitável para o jogo seria de 50 FPS. A partir desse dado, foi possível chegar a um número de 75 pássaros por geração o que nos dá 86 nós e cerca de 50 FPS ao início de cada geração.

7.2 *Rede Neural*

A Rede Neural do jogo foi implementada no modelo Brain que tem como principal método, o think. Este método recebe um array de inputs como parâmetro e retorna verdadeiro se o pássaro deve pular ou falso, quando não.

O modelo pode ser inicializado de duas formas: a primeira é passando com quantos inputs o modelo terá que trabalhar. Uma vez passadas as quantidades de inputs, o modelo cria uma lista de pesos com valores aletórios de -1 até +1. Esta forma é usada pelo modo AI Mode para gerar os pássaros da primeira geração.

A segunda maneira é passando os pesos para o inicializador, assim, o modelo vai salvar estes pesos e usá-los para fazer os cálculos. Deste jeito, podemos salvar quais foram os pesos de um pássaro que obteve um desempenho muito bom e usar este mesmo peso para inicializar um pássaro igualmente bom no modo Versus Mode.

Para garantir que a escolha dos inputs seria feita de maneira correta, uma bateria de testes extensivos foram feitos. Em cada um dos conjuntos de inputs a seguir foram rodados 35 vezes até que o algoritmo genético com a rede neural com os respectivos parâmetros chegasse na vigésima geração. Os resultados foram configurados para aparecer em gráficos de linha para mais fácil visualização.

Primeiramente foi feito um teste com 4 *inputs*: altura do pássaro; altura do cano inferior; altura do cano superior; velocidade vertical do pássaro, que pode ser identificado na Figura 9 como a linha verde. Os primeiros testes já se mostraram positivos visto que foram melhores que o aleatório , representado na Figura 9 como a linha azul, o pássaro entendendo o que eram obstáculos e onde eram suas passagens. Porém, foi possível ver que o pássaro teve dificuldade principalemente quando caía de uma altura grande para um obstáculo mais embaixo, problema dado graças ao parâmetro de velocidade vertical.

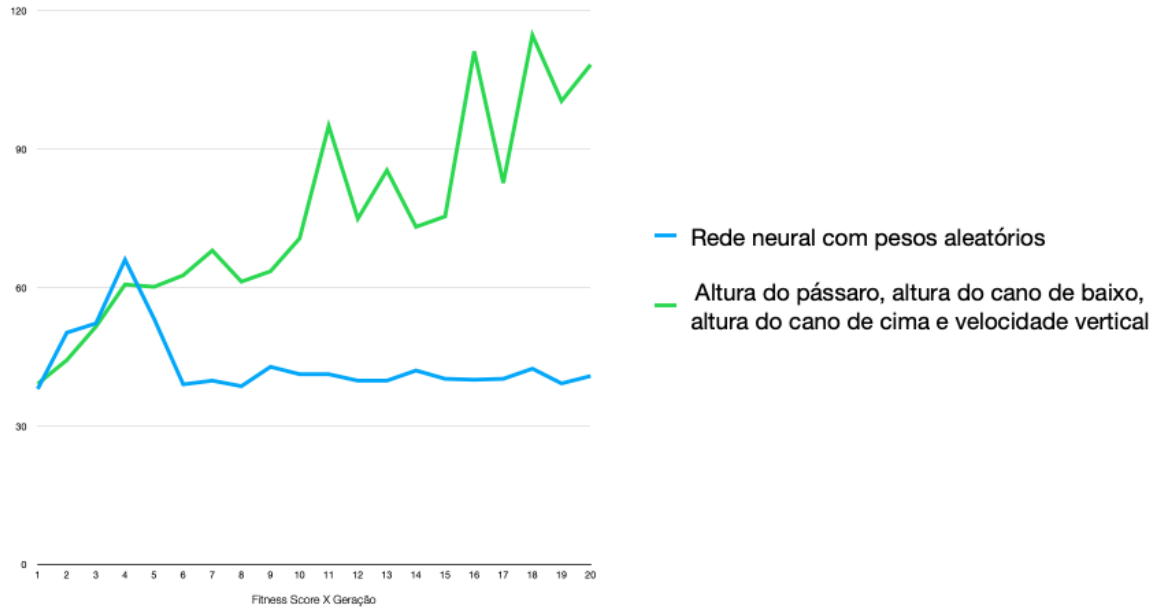

Figura 9: Gráfico de geração no eixo X por Fitness Score no eixo Y. Comparando a rede neural com pesos aleatórios e inputs de altura do pássaro, altura do cano de baixo, altura do cano de cima e velocidade vertical

Para seguir, foi decidido diminuir a quantidade de inputs para o menor número possível para que o algoritmo não se confundisse. Com isso mais dois testes foram feitos, cada um com dois parâmetros foram feitos, o primeiro com: altura do pássaro; altura do cano inferior; e o segundo com altura do pássaro e altura do centro do espaço entre os canos. Seguem os resultados:

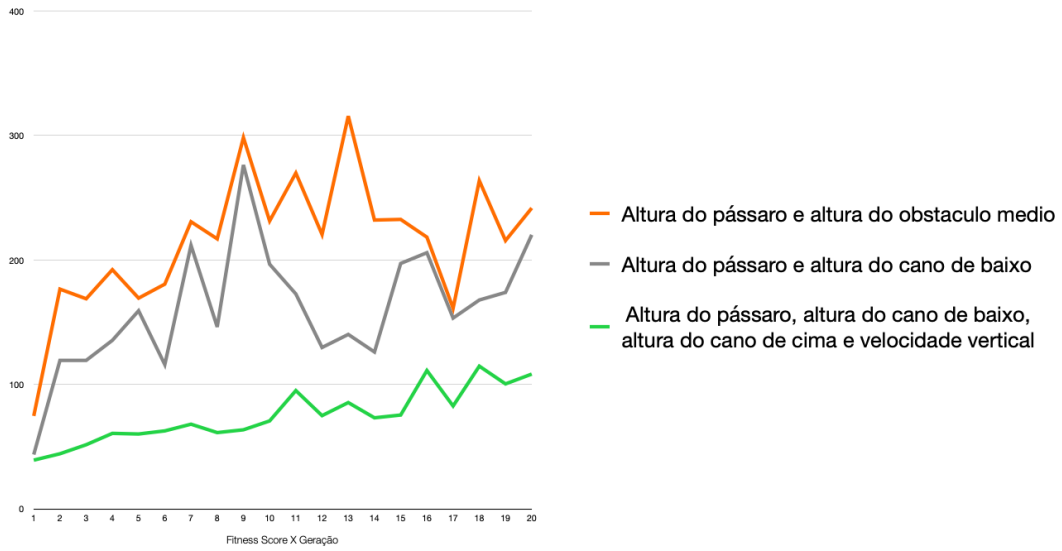

Figura 10: Gráfico de geração no eixo X por Fitness Score no eixo Y. Comparando a rede neural com 4 inputs de altura do pássaro, altura do cano de baixo, altura do cano de cima e velocidade vertical, com a rede neural com 2 inputs de altura do pássaro, altura do cano de baixo e a rede neural com 2 inputs de altutra do pássaro e altura do obstáculo médio.

Como podemos ver no gráfico acima, ambos os novos resultados, representados pelas cores cinza e laranja, apresentam melhorias em comparação com o primeiro, representado na Figura 10 pela cor verde. Ainda assim, não parecia ser o melhor resultado possível uma vez que deu pra notar que o pássaro perdia muitas vezes quando batia no cano superior por não ter este dado tão claro.

Pensando nisso mais um teste foi rodado com os seguintes inputs: Altura do pássaro, altura do cano de baixo, altura do cano de cima e distância do pássaro para o próximo cano. Os resultados desse teste são mostrados com a linha de cor azul escura no gráfico a seguir:

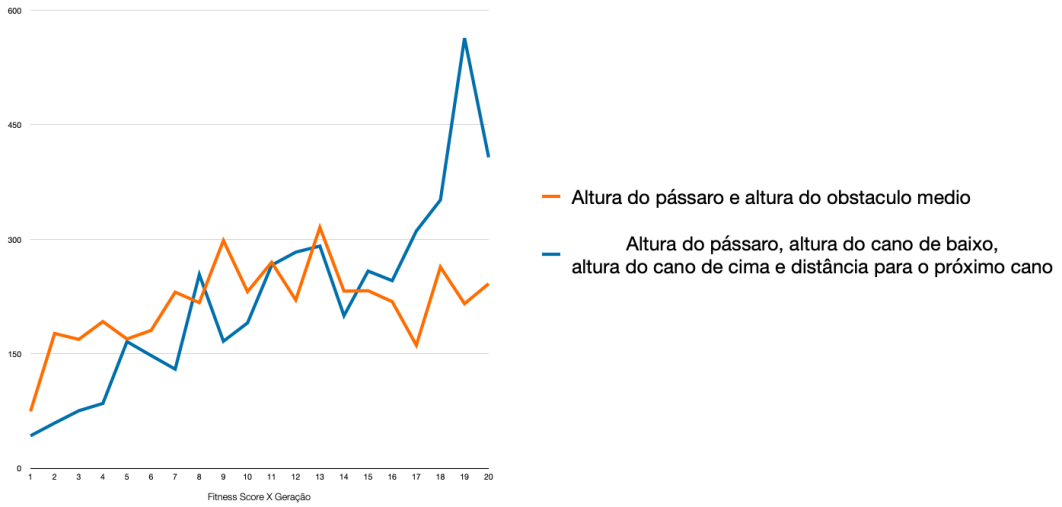

Figura 11: Gráfico de geração no eixo X por Fitness Score no eixo Y. Comparando a rede neural de 2 inputs de altutra do pássaro e altura do obstáculo médio com a rede neural de 4 inputs de altura do pássaro, altura do cano de baixo, altura do cano de cima e distância para o próximo cano.

Estes resultados foram muito positivos pois era possível ver, que mesmo em algumas gerações mais recentes, o pássaro alcançando pontuações elevadas, e morrendo apenas por causa de um bug no momento da criação de obstáculos onde era impossível ultrapassar o obstáculo sem tocar no cano, ou seja morrer. O maior problema desses inputs era a sua alta inconsistência, uma vez que era possível que na terceira geração um pássaro alcançasse os cem pontos contudo, em outras iterações não passando de quatro pontos ao fim das 20 gerações.

Com o intuito de resolver o problema foi feito um *debug* nas gerações que estavam indo bem para ver a situação dos pesos de cada *input*. Foi chegada a conclusão que o *input* de distância entre o pássaro e o cano estava somente tornando mais dificil pro algoritmo de alcançar o resultado ótimo visto que os pássaros com melhor *fitness score* possuíam peso zero neste *input*. Pensando nisso, mais um teste foi rodado com os seguintes parâmetros: Altura do pássaro, altura do cano de baixo, altura do cano de cima; Segue o resultado:

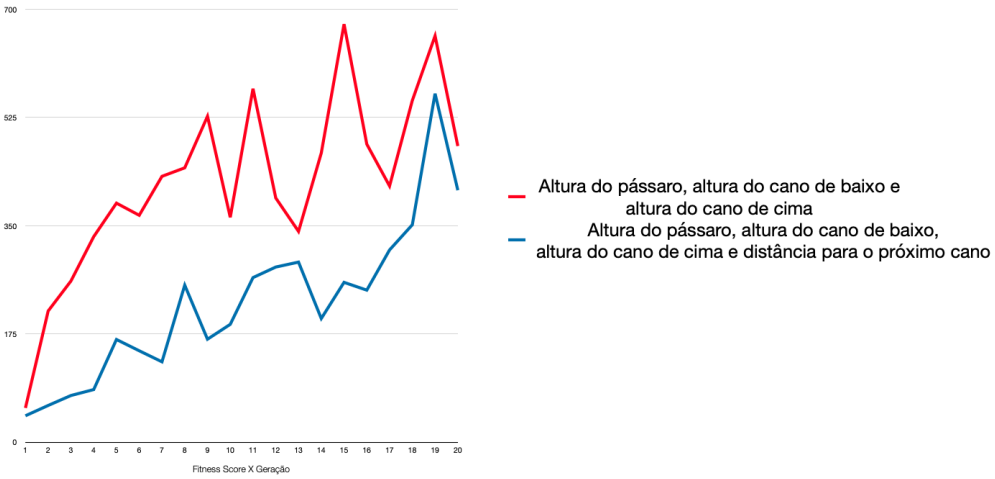

Figura 12: Gráfico de geração no eixo X por Fitness Score no eixo Y. Comparando a rede neural de 4 inputs de altura do pássaro, altura do cano de baixo, altura do cano de cima e distância para o próximo cano com a rede neural de 3 inputs de altura do pássaro, altura do cano de baixo, altura do cano de cima.

Os resultados deste teste estão representados em vermelho neste gráfico e deixam claro que estes inputs conseguem alcançar resultados melhores em menos gerações do que quando comparado com os outros, com estes resultados foi mais fácil decidir de maneira acertiva os inputs corretos para a versão final do projeto.

Com o gráfico a seguir podemos ver a importância de escolher os inputs certos da rede neural e como fazem diferença para o resultado final.

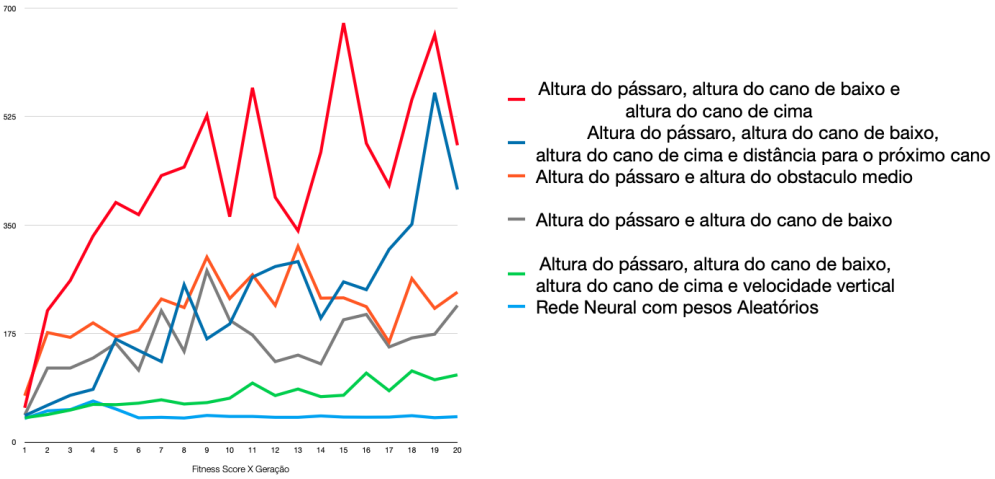

Figura 13: Gráfico de geração no eixo X por Fitness Score no eixo Y. Comparando todos os gráficos apresentados anteriormente.

7.3 *Algoritmo Genético*

A implementação do AI Mode é baseada em algoritmo genético com recompensa. No começo do modo geramos 75 pássaros com seus valores de peso aleatórios em suas redes neurais e iniciamos o jogo com eles. Alguns, logo de cara caem ou saem do mapa, porém alguns outros conseguem chegar no primeiro cano e dependendo, passar do primeiro obstáculo.

Assim que um *AIBird* morre no jogo, atrubuimos um fitness score para ele. No início da implementação, esta pontuação era a mesma da existente no Player Mode, onde a cada vez que um pássaro passa pelo obstáculo ele soma um ponto. Isto era um problema para as primeiras gerações pois pássaros que voavam em direção ao teto ou ao chão do jogo ficavam com a mesma pontuação de um pássaro que conseguia se manter no ar mas não passava pelo cano. Pensando nisso, a implementação do fitness score foi mudado para que fosse equivalente a distância que o pássaro conseguiu alcançar, melhorando bastante a solução pois tinhamos mais claramente o melhor pássaro da geração.

Após todos os agentes de um geração morrerem, uma nova geração é formada, porém dessa vez, a geração é composta por 74 cópias do melhor pássaro da geração anterior de acordo com o fitness score dos agentes. Para que haja um avanço na solução do problema, as cópias dos pássaros passam por um método de mutação, neste método, cada peso é somado de um valor aleatório entre -0.1 e 0.1. Desta forma, um novo pássaro pode surgir sendo melhor ou até mesmo pior que o pássaro modelo da geração anterior.

O gráfico abaixo foi gerado a partir de trinta e cinco iterações do algoritmo genético até alcançar a vigésima geração. Também foi usado um algoritmo com escolha aleatória como comparação. Com o gráfico podemos observar a melhoria do algoritmo genético com o passar das gerações. Além disso, o melhor fitness score obtido em todas as iterações do algoritmo aleatório foi 166 enquanto que o algoritmo genético, em sua melhor iteração conseguiu alcançar uma pontuação de 2920.

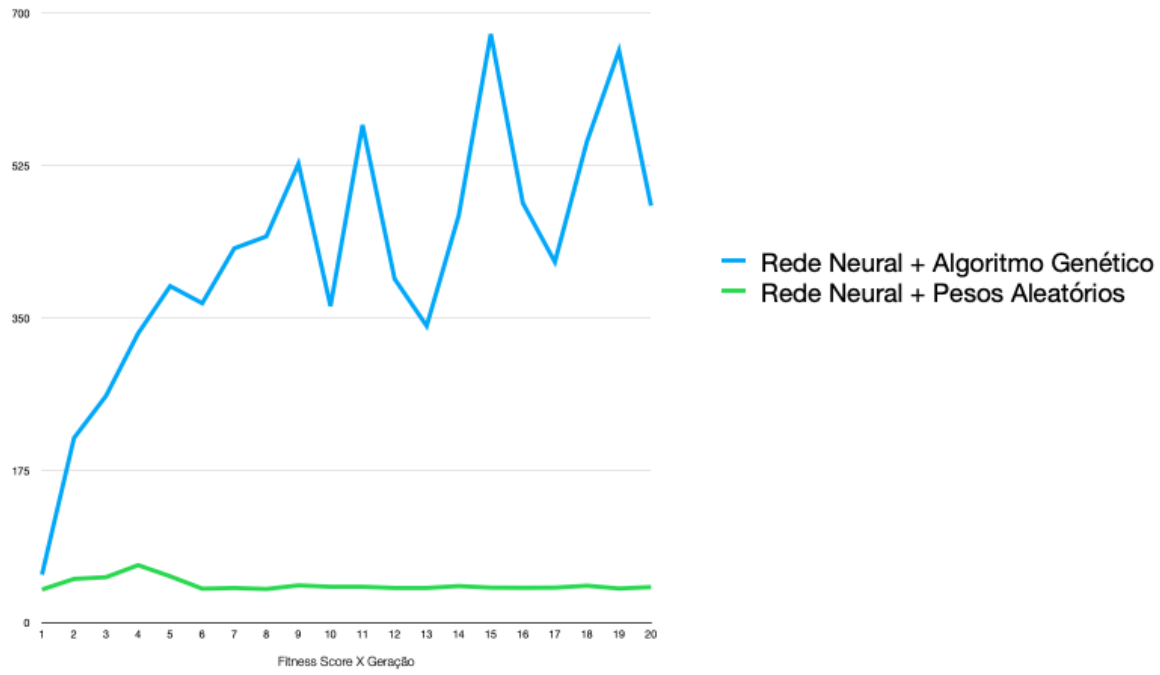

Figura 14: Gráfico de geração no eixo X por Fitness Score no eixo Y. Comparando a rede neural com pesos aleatórios e a rede neural evoluindo com o algoritmo genético.

8 Considerações finais

Ao final do projeto, disponivel no meu GitHub pessoal, vejo que o objetivo de recriar o jogo Flappy Bird com um agente que consiga jogar melhor que um humano foi concluído. Podemos ressaltar que Flappy Bird é um ótimo jogo para o estudo da Inteligência Artificial por ser simples e também desafiador, que fornece um ambiente propício para a contrução destes tipos de algoritmos, sendo possível criar um agente simples que consegue vencê-lo.

8.1 *Melhorias possíveis*

Existem algumas melhorias possíveis na forma que o jogo foi implementado que podem tanto melhorar a experiência de um jogador humano quanto melhorar os resultados no *AI Mode*. Algumas delas são:

- A maneira de criar obstáculos que foi implementada no jogo deve ser melhorada pois ao criar novos obstáculos o jogo perde alguns frames. Isso pode causar frustração do jogador humano em ver a imagem travando sempre que um novo obstáculo nasce mas também pode causar um *AIBird* bem treinado que, naquele momento, não consegue executar o comando de pular pois o jogo está tentando renderizar o novo obstáculo na tela.
- A principal mudança que pode ser realizada é corrigir as situações em que são criados obstáculos impossíveis de serem atravessados sem morrer, isso porque o próximo cano estava ou muito embaixo ou muito em cima em relação ao anterior. Outra consequência relacionada a este *bug* é a perda de um *AIBird* bom, visto que uma vez ele se deparar com o obstáculo impossível, muitos pássaros da geração morrem com o mesmo *fitness score* e não é possível definir qual é o melhor entre eles. Se não fosse por este bug é provável que muitas iterações teriam gerado um pássaro capaz de nunca perder um jogo, como é o caso do print abaixo.

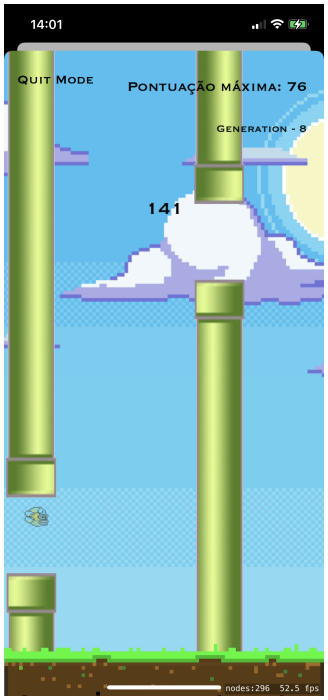

Figura 15: Caso em que é impossível de passar pelo próximo obstáculo

Algo que também seria interessante como melhoria para este projeto é a criação de uma rede neural mais complexa com mais camadas que a atual, podendo também usar outros inputs que não sejam tão alto nível como os usados no presente projeto, passando informações que não foram pré-procesadas, diferente da altura do pássaro e distância do cano usadas para que a rede neural tome a decisão.

Um possível problema presente no algoritmo genético é que ele não tem nenhuma forma de escapar de soluções locais, ou seja, uma vez que ele encontra uma melhor solução de pesos da rede neural dentro da variação máxima usada ao mutar o agente, ele não tem nenhuma forma de procurar outras possibilidades com variação maior. A criação de uma parte de agentes com maior variação de mutação pode ser usada para solucionar este problema. Outra ideia poderia ser relacionar a taxa de variação com o fitness score, aumentando a taxa de variação de mutação uma vez que o fitness score esteja baixo e diminuindo se estiver alto.

Por fim, concluo o presente projeto com sentimento de que pude colocar em prática os aprendizados obtidos ao longo do curso e através de oportunidades ofertadas pela PUC-Rio como a Apple Developer Academy, que me introduziu ao mundo do desenvolvimento iOS.

Referências

[1] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition, 2010.

[2] S.M. Lucas and G. Kendall. Evolutionary computation and games. *IEEE Computational Intelligence Magazine*, 1(1):10–18, 2006.

[3] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017.

[4] Atakan Sahin, Efehan Atici, and Tufan Kumbasar. Type-2 fuzzified flappy bird control system. In *2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1578–1583, 2016.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[6] Kevin Chen. Deep reinforcement learning for flappy bird.

[7] Matthew Piper, Pranav A. Bhounsule, and Krystel K. Castillo-Villar. How to beat flappy bird: A mixed-integer model predictive control approach. 2017.

[8] Gerald Tesauro. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6:215–219, 1994.

[9] David Silver, Richard S. Sutton, and Martin Müller. Reinforcement learning of local shape in the game of go. In Manuela M. Veloso, editor, *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 1053–1058, 2007.

[10] OpenAI, :, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique P. d. O. Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning, 2019.

[11] Oriol Vinyals, Igor Babuschkin, Wojciech Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John Agapiou, Max Jaderberg, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575, 11 2019.

[12] Naveen Appiah. Playing flappybird with deep reinforcement learning. 2016.

[13] Yang Shu, Ludong Sun, Miao Yan, and Zhijie Zhu. Obstacles avoidance with machine learning control methods in flappy birds setting. 2014.

[14] Estéfane GM de Lacerda and ACPLF De Carvalho. Introdução aos algoritmos genéticos. *Sistemas inteligentes: aplicaçoes a recursos hıdricos e ciências ambientais*, 1:99–148, 1999.

[15] Diogo C Lucas. Algoritmos genéticos: uma introdução. *Apostila referente a disciplina de Inteligencia Computacional, Universidade Federal do Rio Grande do Sul, Brasil*, 2002.