Relatório sobre o projeto do Mario AI

Alunos: Alan Felix e Ícaro Prado
Profº: Hendrik Macedo

1. Introdução

O Mario AI Championship foi uma série de competições realizadas entre 2009 e 2012 e que foram baseadas em um clone open source do jogo de plataforma Super Mario Bros. As competições eram disputadas por agentes desenvolvidos pela comunidade e eram avaliados de maneiras diferentes, como o Gameplay Track (onde os agentes eram pontuados por quão longe eles sobreviviam em diversas fases de dificuldades diferentes) e o Turing Test Track (onde o agente que mais se parecesse com um humano jogando vencia).
Posteriormente, o campeonato foi modificado para o Plataformer AI Competition, em 2013, remodelado para o jogo open source SuperTux. Essa mudança foi necessária para evitar problemas com os direitos da Nintendo.

2. Funcionamento do jogo

Os agentes somente podem controlar os movimentos do Mario, assim como um humano jogando, ou seja, os botões direita, esquerda, baixo, pular e atirar/correr. Eles recebem as observações do mundo via um array de 22x22, representando a área ao redor do Mario com algumas variáveis de estado (o Mario se encontra na posição [11,11]). O agente tem apenas alguns comandos disponíveis para executar a cada passo de tempo do jogo (40ms). Os comandos do agente são enviados para o servidor em formato de um array de 5 posições: [backward, forward, crouch, jump, speed/bombs]. Por exemplo, caso o agente quisesse pular para frente e atirar ao mesmo tempo em um passo de tempo, o código em python seria:

```
# pular para frente atirando
def act(self):
    return [0, 1, 0, 1, 1].
```

onde o 0 representa uma tecla não pressionada no “controle”.

Existem duas maneiras de colocar o agente para jogar o jogo: a primeira, utilizando o código em java, passando como argumento o código do seu agente (também era possível utilizar código em python, mas a documentação era bastante escassa e não dava para funcionar). A segunda maneira (que foi a alternativa para quem programa em python) é rodando um servidor que envia e recebe os dados via TCP.

3. Implementação

Para implementar um agente para o projeto da disciplina, primeiramente pensamos em alguns algoritmos possíveis e fizemos uma tabela, listando os prós e contras de cada uma das abordagens, segundo conhecimento adquirido ao longo da disciplina e opinião da dupla, uma vez que parte das técnicas possui um alto grau de empirismo.

Tabela de técnicas para o Mario AI			
Técnicas	**Ideia**	**Pontos fortes**	**Pontos fracos**
Agente reativo	Implementar regras simples de acordo com possíveis estados do ambiente	Fácil de implementar para níveis pequenos	A medida que a dificuldade das fases vai aumentando, fica difícil criar regras para que o agente não morra rápido
Greedy search	Desenvolver um agente que trabalha de forma greedy sempre que encontra moedas no cenário.	A depender do método avaliativo, é uma opção viável, em especial se pegar várias moedas for mais relevante do que finalizar o estágio.	O ideal seria utilizar essa técnica em conjunto com pelo menos uma de busca global, o que gera mais complexidade na implementação.
Busca local (A*)	É possível, fazendo a busca em tempo suficiente para prever o que vai acontecer com o Mario	Mais rápido que o Greedy	Como é em tempo real, é necessário um computador forte para fazer a busca em profundidade o suficiente. Essa limitação pode fazer com que o Mario acabe caindo em um buraco vez ou outra.
Aprendizagem por reforço	Gerar valores de recompensa para moedas, matar inimigos e finalizar a fase, bem como penalidades ao ser acertado por um inimigo ou cair em um buraco.	Se adequa às entradas que o agente recebe do sistema, além de ser adaptável um agente para cada possível método avaliativo (quantidade de moedas acumuladas, velocidade em que o Mario finaliza o estágio ou quantidade de inimigos mortos).	A definição das recompensas é algo empírico, requer muito experimento para garantir que um objetivo secundário encontrado várias vezes não acabe se tornando mais relevante do que o objetivo final.
Árvores de decisão ID3	Usando um conjunto de dados de treinamento, podemos usar uma árvore de decisão usando o algoritmo ID3 para jogar o jogo. Os dados do treinamento podem ser coletados de diversas formas, seja de um humano jogando ou uma máquina que já joga bem. De preferência a máquina, pois com isso é possível acumular dados de treinamento em um tempo menor que manualmente.	Dependendo dos dados de treinamento, o agente será bastante eficiente	Necessária uma grande quantidade de dados de treinamento para que o agente jogue bem.

Após pensar em todos os prós e contras, pensamos em implementar duas técnicas: o algoritmo A* e Árvores de decisão (ID3). A ideia seria criar uma base de dados de treinamento utilizando o A*, assim seria possível acumular dados o suficiente para que a árvore de decisão fosse bastante robusta. Como a árvore de computação do A* em um jogo com a complexidade do Mario se tornaria muito grande em pouco tempo, a tendência seria o agente demorar demais para tomar decisões, ou haveria a necessidade de usar algum tipo de poda, o que tornaria a solução ainda mais complexa, porém, dando tempo para esse agente treinar de forma “offline” e aproveitando seu aprendizado para alimentar uma árvore de decisão, seria possível criar um agente com uma rápida resposta “online” mas com uma boa experiência e que possivelmente se daria bem.

A princípio, buscamos como implementar o A*. Encontramos um código de Robin Baumgarten (https://github.com/jumoel/mario-astar-robinbaumgarten), que inclusive foi o primeiro vencedor do Mario AI Championship em 2009. Infelizmente, o código estava em java e não conseguimos fazer o mesmo rodar. Então, encontramos uma versão do algoritmo A* implementado em python para o Mario AI: (https://github.com/snipplets/MarioAI). Aqui nos deparamos com outro problema: o agente desenvolvido pelo autor possuía métodos com argumentos incompatíveis com as observações que são enviadas pelo servidor. Talvez a versão do Mario AI que ele utilizou foi diferente da que temos disponível atualmente na internet (atualmente a mais recente que encontramos foi a de 2011). Isso nos atrasou e não conseguimos extrair a lógica para poder recriar um A* do zero a tempo da competição do dia 13/03/2018.

Caso conseguíssemos implementar o agente em A*, seria a hora de usar Machine Learning para treinar outro agente, que seguiria uma árvore de decisão (usando o algoritmo ID3), assim como o espanhol Luis Cruz foi capaz de fazer. Podemos ver uma explicação em seu site (http://www.emagix.net/academic/item/mario-ai), porém o mesmo não disponibilizou a codificação. No exemplo dele, foi utilizado um humano jogando (ele mesmo) como base de dados para a árvore de decisão, o que, segundo ele, causou bastantes problemas por alguns fatores: quantidade de dados pequena e insuficiente para o treinamento, a demora para fazer isso manualmente, a quantidade de ruído e o fato dele não ser perfeito ao jogar.

Nossa ideia seria pular totalmente a parte do treino com humanos e usar o A* em nosso favor. Assim teríamos uma árvore de decisão robusta em pouco tempo e um agente capaz de avaliar bem uma boa maneira de jogar.

Devido a esses fatores, de última hora, decidimos fazer uma implementação simples de busca. Utilizamos como base o agente ForwardAgent.py e implementamos alguns comportamentos em cima dele. O primeiro, foi fazer com que o agente pule sempre que um bloco esteja acima dele. No array as posições seriam traduzidas como [12,11] e [13,11]. Por alguma razão, na hora da competição em sala, esse comportamento não ocorreu. O agente ficou apenas pulando sempre. O segundo comportamento que foi implementado foi com relação a obstáculos grandes. Fizemos com que sempre que o agente identifique um obstáculo à sua frente durante um pulo simples, ele aperte o botão de pular novamente (fazendo com que ele pule mais alto do que o pulo convencional). No array, a posição seria a logo à frente do Mario, ou seja, [11,12].

4. Conclusão

Não é fácil implementar um agente que jogue um jogo de computador, principalmente um jogo com tantas variáveis como o Super Mario Bros. Os obstáculos que encontramos para

implementar nosso agente foram grandes como o fato da documentação para o python ser quase inexistente, o servidor TCP não trazer algumas das informações que a versão java do código traz no console e o fato de cada um dos códigos que encontramos para servir de base usar uma versão supostamente diferente e/ou modificada do código do servidor. Apesar dos problemas, contornamos com uma solução mais trivial, mas que nos rendeu o 3º lugar na competição. Aprendemos bastante com essa experiência e esperamos que nossos códigos, juntamente com os da turma, sirvam para turmas seguintes caso o professor deseje fazer novamente esse tipo de competição.