Relatório de desenvolvimento de simuladores para a disciplina de IA para jogos

Macártur de Sousa Carvalho

24, Novembro 2017

Contents

1 Ferramentas para contrução de simuladores

Para a contrução dos simuladores foram analisadas algumas plataformas/*engines* que permitisse a a criação de objetos simulados e a utilização de técnicas de Inteligencia Artificial aprendidos durante a disciplina de Inteligência Artificial para jogos.

Abaixo é apresentado as plataformas analisadas e alguns detalhes sobre elas.

1. **Unity** é uma famosa plataforma para criação de sistemas iterativos em 3D e 2D e apresenta recursos que ajudam equipes de artistas e desenvolvedores a criarem jogos. Esta é uma ferramenta de código fechado, com suporte para windows e possibilita desenvolver sistemas iterativos utilizando as linguagens de scripts *TypeScript* e *C#* e contrução de projetos para para diferentes plataformas, tais como *PS4*, *Linux*, *Windows*, *Mac OSX*, *Android*, *IOS*, *Steam*, *GearVR*, entre outros.

2. **Godot** é uma plataforma que possibilita a criação de sistemas iterativos em 3D e 2D similar a plataforma Unity, no entanto esta é de código aberto e possui suporte para *Windows*, *Mac OSX* e *Linux*. Esta permite a criação de sistemas iterativos utilizando uma linguagem de script propria chamada *GodotScript*, e possibilita criar projetos para diferentes plataformas tais como *Android,Black Barry*, *HTML5*, *Linux*, *Windows* e *Mac OSX*.

3. **Love2d** é uma plataforma para contrução de sistemas iterativos 2D de codigo aberto e com suporte para *Windows*, *Ubuntu* e *Mac OSX*. Essa ferramenta possibilita a contrução de projeto iterativos utilizando a linguagem de *script LUA*, e possibilita criar jogos para *Linux*, *Windows*, *Mac OSX* e *Android*.

A plataforma escolhida para criação dos simuladores aqui apresentados foi a **Godot**, pois essa é uma ferramenta que possui recursos para desenvolvimento de sistemas iterativos em 2D, de código aberto e permite a distribuição do sistema para diferentes plataformas. Para utilizar a plataforma basta acessar o endereço `https://godotengine.org/download` e fazer o download do executável para a plataforma que deseja desenvolver seu projeto. A fígura 1 apresenta a interface gráfica da plataforma escolhida.

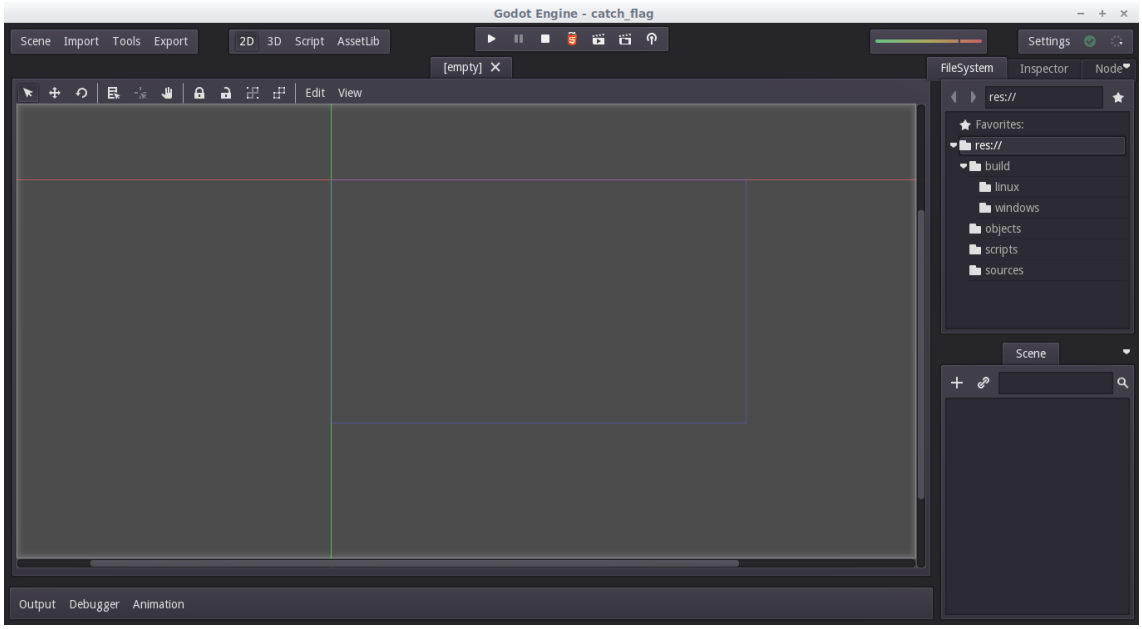

Figure 1: Interface gráfica da plataforma Godot

2 Métodos utilizados

2.1 Autômato célular

2.1.1 Representação do cenário

Para facilitar a simulação de agentes influênciados pelo ambiente foi utilizado um modelo de discretização chamado **autômato celular**, apresentado no capítulo 6.2.8 do livro *Inteligencia artificial para jogos* [1]. Com este modelo o cenário foi todo representado por uma grade finita de duas dimensões com células retângulares, onde cada célula apresenta um número finito de estados e possui cada estado determinado por um conjunto de regras determinísticas. Para a representação computacional dessa grade foi utilizado uma matriz de duas dimensões. Onde cada posição da matriz é representada por um retângulo da grade do cenário.

Com a discretização utilizando o modelo de grade um objeto é posicionado em uma das posições da matriz e um desenho deste objeto é posicionado na tela utilizando a linha e coluna do objeto representado na matriz. O cenário da figura 2 é um exemplo de cenário contruído utilizando grade e possui como matriz a tabela apresentada em 1, onde temos uma chão vazio, pessoa e bloco de parede representados pelos valores 1, 2 e 3 respectivamente.

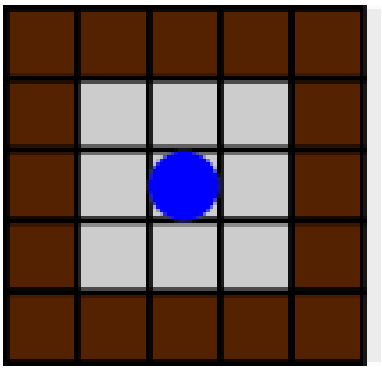

Figure 2: Exemplo de cenário criado utilizando grade.

Table 1: Exemplo de cenário representado por uma matriz

3	3	3	3	3
3	1	1	1	3
3	1	2	1	3
3	1	1	1	3
3	3	3	3	3

[1] https://www.amazon.com.br/Artificial-Intelligence-Games-Ian-Millington/dp/0123747317

2.1.2 Representação de mapa de influência

Com o cenário representado como uma matriz podemos utilizar um **mapa de influência** deste cenário. Mapa de influência é uma representação discreta da influência de cada um dos objetos presentes no cenário sobre as células visinhas.

A fórmula utilizada para calcular a influência sobre cada célula é apresentada no capítulo 6.2.2 do livro texto *Inteligencia artificial para jogos* [1], onde temos:

$$I_d = \frac{I_0}{(1+d)^2}$$

Nesta fórmula o valor de I_d é a influência resultante baseado na distância d e a influência inicial I_0 da célula. A tabela 2 apresenta um exemplo de cálculo de uma matriz de influência em relação a um objeto que possui influência de valor +10 e está localizada no meio da matriz quadrada de tamanho 5.

Table 2: Exemplo de cálculo de matriz de influência de um objeto posicionado em [3,3]

0	0	1	0	0
0	1	2	1	0
1	2	10	2	1
0	1	2	1	0
0	0	1	0	0

2.2 Árvore de decisão

Árvore de decisão é uma técnica utilizada para tomada de decisões, apresentada no capítulo 5.2 do livro *Inteligencia artificial para jogos* [1]. Esta técnica consiste em fazer uma ánalise do conhecimento do que aconteceu ou está acontecendo no cenário e tomar uma decisão baseada nesta ánalise.

Um exemplo de aplicação desta técnica seria, dado um jogo de tiro, a cada instante de tempo utilizar a árvore de decisão apresentada na figura 3.

Supondo que temos um jogador que está escondido e não consegue ver um inimigo próximo e consegue escutar os passos do inimigo irá tomar a decisão de rastejar. Isso irá ocorrer por que durante a tomada de decisão ao executar a árvore de decisão irá verificar prímeiro a pergunta *o inimigo está vísivel ?*, como não existe inimigo vísivel ele irá verificar a pergunta *escuto passos do inimigo ?*, como é possivel escutar os passos do inimigo ele irá tomar a decisão de *rastejar*.

[1] https://www.amazon.com.br/Artificial-Intelligence-Games-Ian-Millington/dp/0123747317

[1] https://www.amazon.com.br/Artificial-Intelligence-Games-Ian-Millington/dp/0123747317

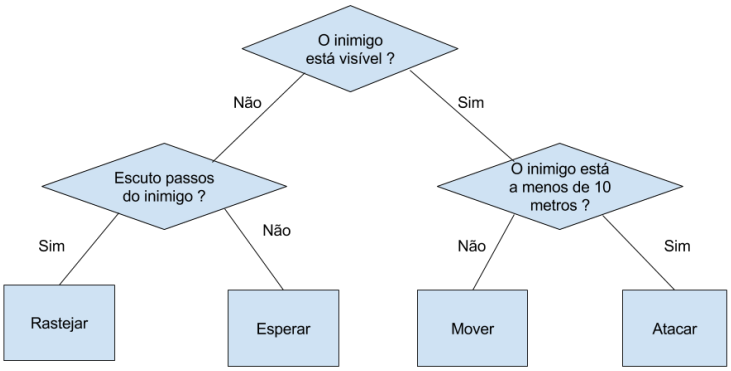

Figure 3: Árvore de Decisão para jogo de tiro

3 Simulador de Estação de Trem

3.1 Público alvo

O projeto de simulador de estação de trem foi desenvolvido para ajudar projetistas e engenheiros no processo de contrução de um projeto de estações de trem, simulando fluxo de pessoas e obstáculos entre a entrada/saída de pessoas na estação e nos vagões de trem.

3.2 Objetivo do sistema

Com o intuito de ajudar projetistas/engenheiros na construção de estações de trem este simulador aqui apresentado foi desenvolvido para auxiliar no projeto de contrução de uma estação de trem, possiblitando a simulação de tráfego de pessoas entre o desembaque e o embarque. Para um projetista de uma estação de trem este projeto pode auxiliar a posicionar obstáculos de maneira que não diminua o fluxo de pessoas que desejam sair e entrar em vagão de trem. Para isso esse projeto tem como principal objetivo permitir que um projetista de estação de trem coloque obstáculos entre as portas de embarque e desembarque de uma estação e verifique qual o comportamento das pessoas na estação de trem depois de colocar um obstáculo, além de definir qual será o melhor fluxo de saída/entrada de pessoas dentro da estação.

3.3 Cenário desenvolvido

Esta seção apresenta uma descrição detalhada do cenário desenvolvido na simulação da estação de trem aqui apresentado. O principal objetivo desta seção é descrever a estação de trem e os objetos representados na simulação.

3.3.1 Descrição do Cenário

A estação de trem representada pelo simulador possui 6 portas, na qual 3 portas se encontram do lado esquerdo da estação e representa a saída/entrada da estação e as outras 3 portas se encontram a direita da estação e representa a entrada/saída de pessoas em vagão de trem. A fígura 4 apresentada abaixo representa uma estação de trem com um fluxo de pessoas saíndo e entrando na estação utilizando as 6 portas e alguns obstáculos entre as portas.

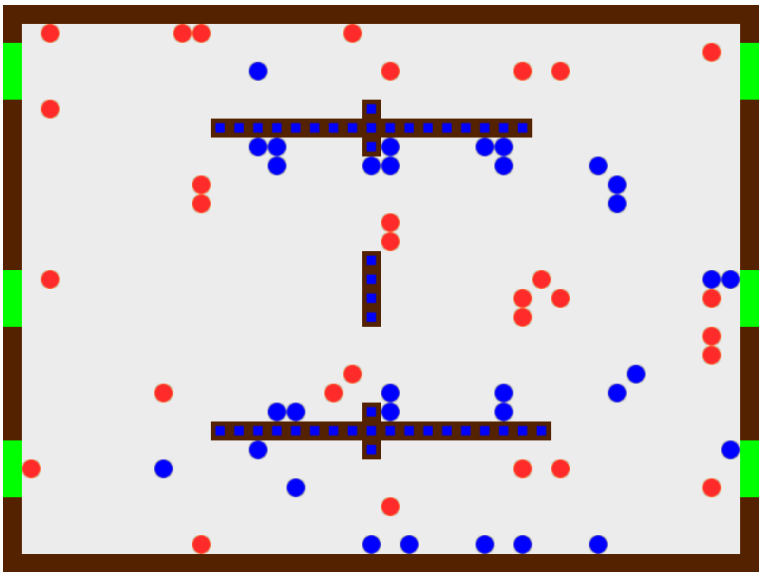

Figure 4: Estação de Trem

3.3.2 Objetos simulados

1. Porta de entrada/saída

   Cada porta é representada por três blocos em verde como apresentado na figura 5. Uma porta é capaz de colocar ou remover uma pessoa da estação. Cada porta localizada do lado esquerdo pode colocar pessoas representadas pelas bolinhas em vermelho e remover pessoas representadas pelas bolinhas em azul, caso a porta esteja localizado a direita ela irá colocar pessoas representadas pelas bolinhas em azul e remover pessoas representadas pelas bolinhas em vermelho.

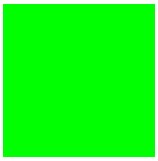

Figure 5: Porta

2. Parede

   Uma parede é composta por um conjunto de blocos representados na fígura 6. Estes blocos são posicionados nas extremidades da estação proíbindo pessoas ultrapassar os limites da estação.

Figure 6: Blocos que compõem uma parede

3. Obstáculo

   Uma obstáculo é representado pela figura 7. Obstáculos podem ser colocados utilizando o botão esquerdo do mouse e retirados utilizando o botão direito. Quando a simulação está rodando não é possível colocar/remover obstáculos da estação.

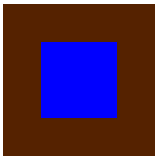

Figure 7: Obstáculo

4. Pessoa

   Pessoas são representadas de duas formas. A fígura 8 representa pessoas que aparecem das portas do lado direito e deslocam-se em direção as portas localizadas no lado esquerdo. A fígura 9 representa pessoas que aparecem das portas do lado esquerdo e deslocam-se em direção as portas localizadas no lado direito.

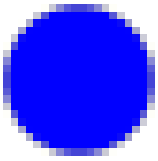

Figure 8: Pessoa tipo 1

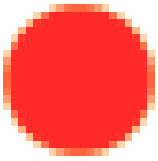

Figure 9: Pessoa tipo 2

3.4 Descrições técnicas

3.4.1 Simulação do cenário

Neste simulador foi utilizado apenas a técnica de discretização com o modelo de autômato célular e utilização de mapa de influência como visto no capítulo 2.1.

Cada porta da esquerda possui um valor de influência igual a 10000, e cada porta da direita possui um valor de influência igual a -10000. Quando calculado as influências das células na matriz de influência é criado um campo de valores que faz com que os maiores valores são colocados mais próximos das portas da esquerda e os menores valores são colocados mais próximos das portas da esquerda. O valor máximo e mínimo de uma célula encontrada no cenário é em cima de uma porta. O valor da influência é alto para evitar com que tenha células visinhas com valores repetidos.

Todos os agentes possuem influência de valor 0 e os blocos das paredes e obstáculos possuem influência de valor 10, para que alguns agentes não chegem perto e outros cheguem.

3.4.2 Simulação de agentes com movimento

Para simular a movimentação de agentes foi utilizado apenas um mapa de influência para todo o cenário. Quando um agente precisa ser movimentado no

método de atualização este calcula qual a melhor opção de se movimentar baseados nos valores das células visinhas.

A tabela 3 apresenta um exemplo das matrizes visinhas de um determinado agente que está localizado no meio da matriz menor. Se este agente tende a ir para uma célula com maior influência então este irá para a célula cujo valor é igual a 12. Caso este esteja localizado no meio e tende a ir para as células com menor influência este irá decidir ir para a célula cujo valor é igual a 5.

Table 3: Exemplo de movimentação de um agente

8	6	5
10	8	6
12	10	8

Desta maneira como as portas possuem os maiores e menores valores de influência os agentes que tendem se movimentar para as células com maior influência irão para as portas da esquerda, e os agentes que tendem a se movimentar para as células com menor influência irão para as portas da direita. Com isso as pessoas representada pelas fíguras 8 e 9 ao serem colocadas no cenário irão se movimentar para as portas de entrada/saída da estação.

3.5 Descrição do sistema

Este sistema foi desenvolvido utilizando a plataforma de desenvolvimento de jogos **Godot**, esta ferramenta possui o recurso de criação de cenários em perspectiva 2D. Os arquivos do projeto aqui desenvolvido se encotram no github como mostado na seção 22.

Inicialmente foram criados cenários utilizando a interface gráfica da plataforma **Godot** . Todos os objetos são vistos como cenários. E existe um cenário principal que é executado com a execução do projeto. Este cenário principal chama todos os outros cenários através da criação de cenários utilizando a linguagem de script **Godot**. Assim cada um dos objetos simulados apresentados na seção 3.3.2 são criados como uma cena.

Após a criação de todos os cenários, o script chamado *world.gd* foi criado dentro da pasta *scripts* e foi adicionado toda a lógica do jogo. Neste script apresenta primeiramente as contantes com o carregamento dos cenários encontrados no diretório *objetos*.

Dentro deste projeto foram criadas as classes apresentadas abaixo:

1. **Game**: esta é a classe que possui um objeto que contém o menu lateral do jogo para iniciar o jogo, remover o objetos colocados e parar o jogo. No método de atualização está apenas chama o metodo de atualização do mapa depois que verificou todos as opções escolhidas no menu.

2. **Map**: esta é a classe principal, onde cria os objetos do cenário e coloca cada objeto no seu lugar ínicial. Nesta classe está contido o mapa de influência do cenário e um mapa de posicionamento dos objetos no cenário.

A cada frame de execução o método de atualização é chamado para atualizar o cenário, este percorre todos os objetos presentes na matriz de posições do cenário e executa o método de atualização de cada um dos objetos simulados do cenário.

3. **Block**: esta classe é utilizada para impedir que as pessoas saiam pelas extremidades do cenário. Se um bloco é colocado fora das extremidades do cenário este muda de textura e é representado como um obstáculo.

4. **Door**: esta classe é utilizada para representar uma porta e possui métodos para criar e remover pessoas do cenário e adicionar influência no mapa de influência.

5. **Agent**: esta classe representa as pessoas que são colocadas no cenário. Ela possui métodos para criação e remoção de sua influência, método de instanciação para colocar os valores padrões das pessoas, um método para remover a pessoa do cenário e um método que atualização a posição da pessoa de acordo com o mapa de influência do jogo.

Depois temos as definições de funções para iniciar, atualizar e receber os sinais de entrada do simulador.

1. *_ready*: é a função principal do simulador na qual é executado quando o cenário é criado. Este método registra que será utiliado os métodos *_process* e *_input* e cria uma instância da classe principal Game.

2. *_process*: é a função que é executada a cada frame e possui apenas a execução do método update do objeto que da classe Game.

3. *_input*: é a função que é executado cada vez que um botão do mouse é clicado. Nesta função se o simulador não estiver rodando a cada clique do mouse irá criar ou remover um obstáculo do cenário.

Com isso ao executar o simulador o cenário inicial é executado e cria a janela mostrada na figura 10. No canto direito existe um menu que apresenta itens abaixo.

1. **Painel de status**: apresenta um texto que diz se o simulador está rodando ou está parado.

2. **Left Door Speed**: É um botão de opções para controlar a velocidade de criação das pessoas que são criadas nas portas da esquerda.

3. **Right Door Speed**: É um botão de opções para controlar a velocidade de criação das pessoas que são criadas nas portas da direita.

4. **Start**: Inicia a simulação e modifica o painel de status para *Running ...* .

5. **Stop**: Para o simulador e modifica o painel de status para *Stopped*.

6. **Clear Agents**: É um botão para remove todos as pessoas que estão no simulador.

7. **Clear Train Station**: É um botão para remove todos obstáculos colocados com o botão esquerdo do *mouse*.

Figure 10: Interface do simulador de estação de trem.

3.6 Próximos passos

Como próximos passos de evolução do simulador temos as seguintes opções:

1. **Pesquisa de campo**: pesquisa de campo para melhorar a simulação do comportamento das pessoas presentes na estação.

2. **Gráfico de estatisticas**: criar um gráfico de fluxo de pessoas dentro da estação, onde cada retângulo da grade do cenário apresenta a quantidade de pessoas que passou por aquele retângulo.

4 Simulador de Jogo Competitivo

4.1 Público alvo

Pessoas que queriam apresentar o jogo competitivo pega bandeira para pessoas que não conhecem as regras, cenário e objetivo.

4.2 Objetivo do sistema

Este sistema foi desenvolvido pensando em demostrar como o jogo pega bandeira é jogado. Com este simulador aqui descrito é possível criar dois times, com quantidade de pessoas variadas em cada time e simular uma partida do jogo.

4.3 Cenário Desenvolvido

Esta seção apresenta uma descrição detalhada do cenário desenvolvido na simulação do jogo pega bandeira aqui apresentado. O principal objetivo desta seção é descrever como o jogo funciona e os objetos representados no cenário.

4.3.1 Descrição do jogo Pega bandeira

O jogo pega bandeira é uma brincadeira na qual os participantes são divididos em dois times que ocupam um campo dividido em duas partes iguais. Cada time possui um local onde existe uma quantidade de bandeiras guardadas e que fica mais distante do campo do time adversário, para dificultar que os membros do time adversário pegue a bandeira.

O objetivo do jogo é atravessar o campo do time adversário e pegar uma bandeira sem ser pego. Cada jogador pode pegar apenas uma bandeira. Se um jogador está no time adversário e for tocado por um dos mebros do time adversário deve ficar parado, congelado, no território inimigo. Um jogador congelado não pode se mover até que um membro da equipe aliada descongelado o toque. Se um jogar é congelado e este estiver com a bandeira, a bandeira que estiver com o jogador deve retornar para o local onde foi retirada.

Um exemplo de cenário desta brincadeira é apresentado na fígura 11.

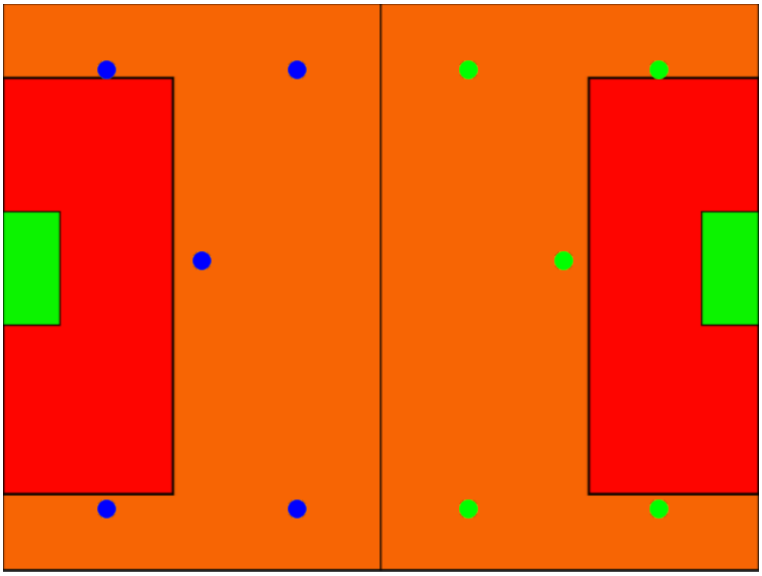

Figure 11: Cenário do jogo Pega bandeira

4.3.2 Objetos representados

1. **Jogador da Esquerda**

   Cada jogador é representado por 3 tipos de figuras diferentes. A figura 15 representa o jogador descongelado. A figura 16 representa um jogador que foi tocado por um jogador do lado direito. E por fim a figura 17 representa um jogador que atravessou o campo adversário e está segurando uma bandeira.

Figure 12: Jogador do time azul descongelado

Figure 13: Jogador do time azul congelado

Figure 14: Jogador do time azul com uma bandeira

2. **Jogador da direita**

   Cada jogador é representado por 3 tipos de figuras diferentes. A figura 15 representa o jogador descongelado.A figura 16 representa um jogador que foi tocado por um jogador do lado esquerdo. E por fim a figura 17 representa um jogador que atravessou o campo adversário é pegou uma bandeira.

Figure 15: Jogador do time verde descongelado

Figure 16: Jogador do time verde congelado

Figure 17: Jogador do time verde com uma bandeira

4.4 Cenário Desenvolvido

4.4.1 Simulação do cenário

Para simular o cenário foi utilizado o modelo de discretização apresentado no capítulo 5, onde o cenário apresentado é dividido e forma uma grade com vários retângulos.

Utilizando esta grade foi criado uma matriz de posicionamento de objetos, 6 matrizes de influência e uma matriz para indicar áreas seguras do cenário.

A matriz de posicionamento de objetos contém apenas a referência de cada jogador presente no cenário. Com estas referências foi criado uma matriz para representar as áreas seguras para cada jogar e foram criadas 6 matrizes de influências.

A matriz de área segura utiliza os números abaixo para indicar qual a área que o jogador se encontra.

1. Indica que está na área não segura do time azul.
2. Indica que está na área não segura do time verde.
3. Indica que está na área segura do time azul.
4. Indica que está na área segura do time verde.
5. Indica que está na área que contém bandeira pertencente ao time azul.
6. Indica que está na área que contém bandeira pertencente ao time verde.

As 6 matriz de influências presentes estão descritas abaixo.

1. **blue**: Contém a influência de cada jogador do time azul e que não estão congelados.
2. **green**: Contém a influência de cada jogador do time verde e que não estão congelados.
3. **blue_frozen**: Contém a influência de jogadores do time azul que estão congelados.
4. **green_frozen**: Contém a influência de jogadores do time verdeque estão congelados.
5. **blue_with_flag**: Contém a influência de jogadores do time azul que possuem uma bandeira.
6. **green_with_flag**: Contém a influência de jogadores do time verde que possuem uma bandeira.

4.4.2 Simulação de agentes com movimento

Para simular a movimentação dos jogadores presentes no cenário foi utilizado o modelo de discretização do cenário com as matrizes de influências apresentadas no capítulo 18 e o método de árvore de decisão mostrado no capítulo 6.

A cada instante que é processado a inteligencia de um agente a árvore de decisão representada pela fígura 18 é executada e o agente toma a decisão de como ele irá se mover. Se simulador está parado ou o jogador está congelado a árvore de decisão não é executada.

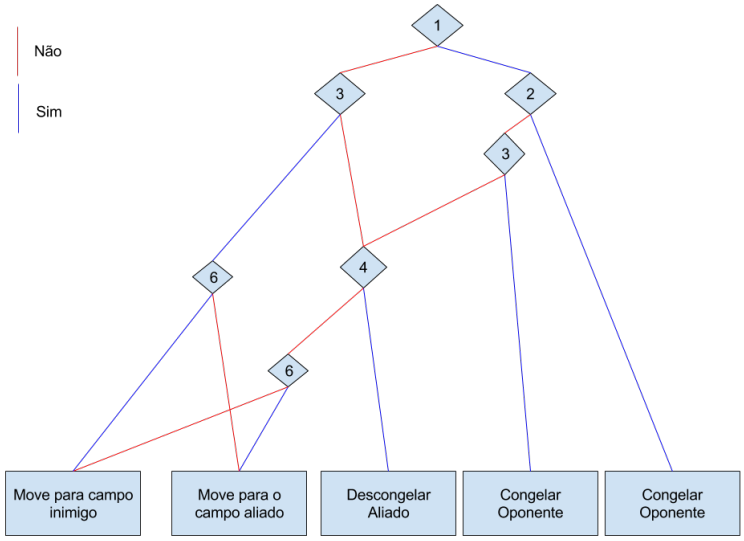

Figure 18: Árvore de Decisão pega bandeira

4.5 Descrição do sistema

Este sistema foi desenvolvido utilizando a plataforma de denseolvimento de jogos **Godot**, esta ferramenta possui o recurso de criação de cenários em perspectiva 2D. Os arquivos do projeto aqui desenvolvido se encontram no github como mostrado na seção 22.

Inicialmente foram criados cenários utilizando a interface gráfica da plataforma **Godot** . Todos os objetos são vistos como cenários. E existe um cenário principal que é executado com a execução do projeto. Este cenário principal chama todos os outros cenários através da criação de cenários utilizando a linguagem de script **Godot**. Assim cada um dos objetos simulados apresentados na seção 4.3.2 são criados como uma cena.

Após a criação de todos os cenários, o script chamado *world.gd* foi criado dentro da pasta *scripts* e foi adicionado toda a lógica do jogo. Neste script apresenta primeiramente as contantes com o carregamento dos cenários encontrados

no diretório *objetos* e texturas encontradas no diretório *sources*.

Em seguida foram declaradas as classes apresentadas abaixo:

1. **Player**: é a classe que representa um jogador, ela possui métodos de instanciação, atualização, remoção, adicionar e remover influência dos mapas de influência, congelar e descongerar, pegar e remover uma bandeira de um jogador. O método de atualização encontra-se toda a lógica de utilização da árvore de decisão e decide qual a próxima célula que o jogador irá se mover.

2. **IA**: é a classe que contém os mapas de influência, as áreas seguras do jogo e um método para devolver mapas de influência com aliados, aliados congelados, oponentes, oponentes congelados e oponentes com bandeira.

3. **Field**: é a classe responsável para montar a matriz de posição dos jogadores no mapa e pode mover, criar e atualizar os jogadores do cenário e possue métodos de verificação para não deixar que um jogador saía do cenário.

4. **Menu**: é a classe responsável para verificar o *status* e armazenar o status do menu. O menu armazena possue botões para criar jogadores, parar e rodar o simualdor, armazenar se o simulador está rodando ou parado, exibir a pontuação de cada time com a quantidade de bandeiras capturada e mostra quando um time ganha se a quantidade de bandeiras capturada é maior que o valor da opção da quantidade de bandeiras máxima permitida.

5. **Game**: é a classe principal do simulador que possui atributos da classe Menu e Field. Quando o método atualizar é executado o menu é atualizado e verifica no menu se o *status* do jogo está como rodando, caso esteja irá executar o método update do atributo do tipo Field.

Depois de criado as classes as funções abaixo foram criadas para iniciar, atualizar e receber os sinais de entrada do simulador.

1. *_ready* é a função principal do simulador na qual é executado quando o cenário é criado. Este método registra que será utiliado os métodos *_process* e *_input* e cria uma instância da classe principal Game.

2. *_process* é a que é executada a cada frame e possui apenas a execução do método update do objeto que da classe Game.

3. *_input*: é a função que é executado cada vez que um botão do mouse é clicado. Nesta função se o simulador não estiver rodando a cada clique do mouse irá criar ou remover um jogador do cenário.

Com isso ao executar o simulador o cenário inicial é criado e uma é janela exibida, como mostrado na figura 19. No canto direito existe um menu que apresenta itens abaixo.

1. **Painel de status**: Apresenta um texto que diz se o simulador está rodando ou está parado.
2. **Start**: Inicia a simulação e modifica o painel de status para *Running ....*
3. **Stop**: Para o simulador e modifica o painel de status para *Stopped ....*
4. **Create default players**: É um botão para criar jogadores em posições pré-definidas.
5. **Seleção de time**: Este botão é utilizado para selecionar o time que vai ser colocado ao clicar com o botão esquerdo do mouse. Se o botão estiver selecionado em "Blue" irá colocar jogadores do time azul, caso esteja selecionado em "Green" irá colocar jogadores do time verde.
6. **Reset**: É um botão para remove todos jogadores do cenário e modificar a pontuação dos dois times para zero.
7. **Edite Mode**: é um botão para entrar em modo de edição e permitir colocar jogadores no cenário.

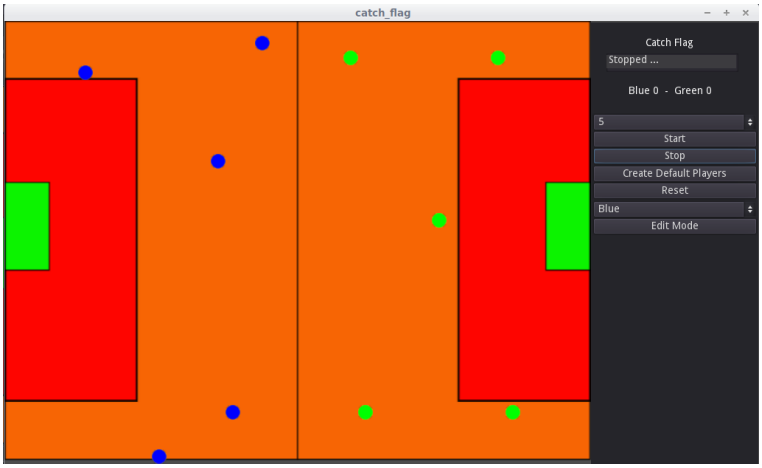

Figure 19: Interface do simulador do jogo conmpetitivo pega bandeira.

4.6 Próximos passos

Como próximos passos de evolução do simulador devemos pesquisar e perguntar para professor experientes no jogo pega bandeira quais são as melhores estratégias para conseguir um bom desempenho no jogo pega bandeira.

5 Github

5.1 Estrutura dos repositórios

Todo os simuladores apresentados neste relatório se encontram no github e possui a estrutura de diretórios apresentadas abaixo.

1. **build**: Diretório onde se encontra arquivos que podem ser executados em sistemas operacionais windows e linux.
2. **source**: Diretório onde se encontra imagens utilizadas no projeto.
3. **objects**: Diretório onde se encontra todos os cenário utilizado no projeto. Todos os objetos simulados encontrados na seção 10 se encontram nesta pasta.
4. **scripts**: Diretório que possui apenas um arquivo chamado *world.gd* que contém todo o código do simulador escrito em Godot Script. Neste arquivo a cena principal chama todos cria todos os objetos necessários.
5. **arquivos do diretório raiz**: São arquivos de configuração que foram editados apenas pela plataforma **Godot** durante a criação das cenas e personalização de exportação e executação dentro da plataforma. Os arquivos encontrados na raiz são (engine.cfg, export.cfg, icon.png icon.png.flags)

5.2 Execução dos projetos

Os projetos aqui apresentados estão disponíveis no github nos endereços mostrado abaixo.

1. **Estação de Trem**: `https://github.com/macartur/train_station`
2. **Pega Bandeira**: `https://github.com/macartur/catch_flag`

Para executar cada um dos projetos basta entrar no diretório build e entrar no diretório windows ou linux dependendo do sistema operaciona que o simulador será executado. Dentro de cada um destes diretórios encontra-se um executável na arquitetura 32 bits para cada um dos dois sistemas operacionais.