



INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA (ISEL)

DEPARTAMENTO DE ENGENHARIA ELETRÓNICA E DE
TELECOMUNICAÇÕES E COMPUTADORES (DEETC)

LEIM

LICENCIATURA EM ENGENHARIA INFORMÁTICA E MULTIMÉDIA
INTERAÇÃO EM AMBIENTES VIRTUAIS

Projeto Final

Miguel Silvestre (45101)

Pedro Dias (45170)

Docente

Professor Arnaldo Abrantes

Julho, 2022

Índice

Índice	i
Lista de Tabelas	iii
Lista de Figuras	v
1 Introdução	1
2 Ambiente	3
2.1 Terreno	3
2.1.1 Ruído de Perlin	3
2.1.2 Algoritmo Flood Fill	4
2.2 Blocos	4
2.3 Skybox	4
3 Agentes	5
3.1 Sistema Natural	5
3.2 ML-Agents	5
3.3 Ficheiro de Configuração	6
4 Interface	9
4.1 Menu	9
4.2 Lifebar	9
5 Interação	11
5.1 Rato e Teclado	11
5.2 Microfone	11
5.3 Som	12

6 Conclusão	13
Bibliografia	15

Lista de Tabelas

Lista de Figuras

3.1	Ficheiro de Configuração	6
-----	------------------------------------	---

Capítulo 1

Introdução

Este trabalho visa criar um jogo em Unity [Technologies, 2005], utilizando os conhecimentos adquiridos ao longo do semestre, nomeadamente geração procedimental, aprendizagem automática e interação Pessoa-Máquina.

O projeto consiste na criação de um mundo Minecraft [Studios, 2011], onde agentes são treinados a desempenhar determinadas tarefas e o jogador consegue interagir com o ambiente, modificando o curso dos acontecimentos.

Este documento está organizado da seguinte forma:

TODOTODOTODO esperar final e/ou confirmação led TODOTODO-TODOTODO

Capítulo 2

Ambiente

Neste capítulo será ilustrado a forma como o terreno é criado, os tipos de blocos existentes no jogo e o aspeto com a “Skybox”.

2.1 Terreno

A construção do terreno deste projeto é semelhante à do primeiro trabalho prático. Para ser possível termos um mundo infinito, é necessário este estar otimizado, de forma a diminuir o peso computacional. Assim, um mundo é criado continuamente em torno do agente à medida que este se movimenta e por conseguinte, parte do mundo é “destruído” quando este se afasta consideravelmente.

Como dito anteriormente, este projeto visa a criação de um mundo *Minecraft* [Studios, 2011], ou seja um mundo baseado em cubos. Neste ambiente, existem faces que não são visíveis ao jogador, mas que ao serem criadas e renderizadas tornam a execução mais lenta. Assim, de forma a reduzir o número de triângulos, vértices, etc, e melhorarmos o desempenho, não processámos estas faces.

2.1.1 Ruído de Perlin

O ruído de Perlin [Perlin,] é uma técnica muito usada em jogos e utiliza uma série de números parcialmente aleatórios com o objetivo de imitar objetos naturais, como o sol, nuvens, animações, terrenos, entre outros, e permite o controlo de elementos em pequena e grande escala. Utilizamos este ruído

de Perlin na nivelção do nosso terreno, ou seja quanto maior for a sua regularização, mais nivelado ficará o terreno.

TODO TODO TODO TODO TODO TODO TODO TODO TODO
TODO

METER VALOR QUE USAMOS E PORQUE - ainda não está bem definido so yea

TODO TODO TODO TODO TODO TODO TODO TODO TODO
TODO

2.1.2 Algoritmo Flood Fill

De modo a ser possível construir os diferentes tipos de blocos, utilizamos o algoritmo de Flood Fill [Flo,] que os desenha recursivamente. Em termos gerais, este algoritmo determina a área conectada num nó, dado em um vetor multi-dimensional de nós. Isto possibilita-nos a gerar mais do que um bloco do mesmo tipo, criando assim blocos vizinhos.

2.2 Blocos

tipos de blocos (grass, dirt, stone etc)

mostrar também texturas e materiais

2.3 Skybox

mostrar aspeto final da skybox

Capítulo 3

Agentes

Aqui são identificados os agentes autónomos do projeto e a forma como foram treinados.

3.1 Sistema Natural

Neste projeto, o jogador tem como objetivo manter o funcionamento do sistema natural dos animais.

Existem dois agentes autónomos: lobos e ovelhas. Uma das funcionalidades do jogador consiste em alimentar as ovelhas, mantendo-as vivas. Por sua vez, os lobos procuram ovelhas como fonte de alimento. A vida destes agentes vai diminuindo com o tempo, podendo inclusive chegar ao fim caso não se alimentem. Da mesma forma, a sua vida é prolongada quando se alimentam.

Se o jogador permitir que todas as ovelhas faleçam, os lobos irão morrer também como consequência. É necessário assim manter este equilíbrio natural no maior espaço de tempo possível.

TODOTODOTODOTODOTODO se se reproduzem, spawn com o tempo etc TODOTODOTODOTODOTODO

3.2 ML-Agents

O Unity Machine Learning Agents Toolkit (ML-Agents) [Unity,] consiste num projeto de código aberto, onde jogos e simulações podem servir como ambientes para efetuar o treino de agentes inteligentes.

Os agentes autónomos foram assim treinados com esta ferramenta. Recorrendo a uma aprendizagem por reforço, os agentes conseguem identificar (através de Raycasts) e consumir a sua respetiva fonte de alimentação - lobos comem ovelhas, e estas comida).

3.3 Ficheiro de Configuração

De modo a treinar um agente, é necessário um ficheiro “.yaml” de configuração, onde é possível definir inúmeros parâmetros diferentes. O ficheiro utilizado encontra-se na figura 3.1.

```
behaviors:
  wolfs:
    trainer_type: ppo
    hyperparameters:
      batch_size: 1024
      buffer_size: 10240
      learning_rate: 0.0003
      beta: 0.005
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 256
      num_layers: 1
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
    keep_checkpoints: 5
    max_steps: 2000000
    time_horizon: 64
    summary_freq: 10000
|
```

Figura 3.1: Ficheiro de Configuração

Devido ao número total de parâmetros possíveis definir ser muito elevado,

iremos dar uma breve explicação dos que foram utilizados neste trabalho:

- batch size - número de experiências de cada iteração.
- buffer size - número de experiências colecionadas antes de atualizar o modelo;
- learning rate - corresponde à taxa de aprendizagem;
- Beta - corresponde à força de regularização entrópico. Quanto maior for este valor, mais ações aleatórias são tomadas;
- Epsilon - influencia a rapidez com que a política evolui durante o treino. Estamos a utilizar o valor por omissão 0.2;
- Lambd - parâmetro de regularização com valor por defeito de 0.95. Significa o quão um agente depende dos valores atuais. Se este parâmetro possuir um valor baixo, significa que dependerá mais, enquanto que valores altos fazem com que dependa mais nas recompensas recebidas no ambiente
- Num epoch - também o valor por defeito 3, corresponde ao número de passagens a serem feitas pelo buffer ao realizar a otimização de descida de gradiente
- Learning rate schedule - para o nosso tipo de treino (ppo) o valor de defeito é linear. Este decrementa o learning rate de forma linear, chegando a 0 no “max steps”;
- Normalize - valor booleano que diz se a normalização é aplicada ao vetor de observações de entrada;
- Hidden units - o range varia tipicamente entre 32 e 512, no nosso caso estamos a usar o valor 128 que é o valor por omissão. Este, corresponde ao número de unidades que estão completamente conectadas à camada da rede neural;
- Num layers - número de camadas escondidas na rede neural (valor por defeito é 2);
- Gamma - fator de desconto de recompensas futuras a vir do ambiente;

- Strength - fator a ser multiplicado pela recompensa dada pelo ambiente;
- keep checkpoints - número máximo de “pontos de controle” de modelos a manter. Estamos a usar o valor por defeito (5);
- Max steps - número total de passos que devem ser tomados no ambiente antes do treino terminar;
- Time horizon - Estamos a usar o valor por defeito (64). Significa o número de passos de experiência que cada agente coleciona antes de adicionar ao buffer;
- Summary freq - número de experiências que necessitam ser coletadas antes de gerar e mostrar as estatísticas de treino;

Capítulo 4

Interface

Este capítulo ilustra a interface do jogo.

4.1 Menu

mostrar menu(s)

4.2 Lifebar

mostrar lifebars dos animais e/ou outras (acho que queres para o jogador também)

Capítulo 5

Interação

Este capítulo trata a forma como o jogador interage com o ambiente, nomeadamente recorrendo ao teclado e microfone.

5.1 Rato e Teclado

Tendo como objetivo manter a “vida animal” do ambiente, o jogador consegue criar e destruir cubos, premindo os botões dos lados direito e esquerdo do rato, respetivamente. Assim, consegue controlar mais facilmente o posicionamento destes agentes autónomos.

Apesar dos lobos serem independentes, isto é, conseguem “caçar” a sua presa sem necessidade de intervenção do jogador, as ovelhas não o são, estando dependentes da intervenção humana para se manter vivas. Para as alimentar, o jogador necessita premir a tecla F do teclado, fazendo comida aparecer e atraindo ovelhas.

TODO TODO TODO TODO TODO

CASO CRIEMOS MAIS

TODO TODO TODO TODO TODO

5.2 Microfone

ainda não temos definido, mas ya é obrigatório

5.3 Som

meter aqui música ambiente e efeitos sonoros

Capítulo 6

Conclusão

Este projeto final agregou todos os temas lecionados ao longo do semestre, nomeadamente geração procedimental com a construção de um mundo *Minecraft* [Studios, 2011], aprendizagem automática onde agentes foram treinados a se alimentarem para sobreviver, e interação Pessoa-Máquina de modo a ser possível ao utilizador interagir e alterar o curso dos acontecimentos.

Assim, foi possível consolidar os conhecimentos adquiridos ao longo das aulas, que já tinham sido postos em prática nos mini-projetos, em algo mais complexo e onde, havendo inúmeras funcionalidades adicionais que possam ser criadas, seja possível explorar estes conceitos livremente.

TODO TODO TODO TODO TODO TODO TODO TODO TODO

não sei bem o que escrever mais, o segundo parágrafo já foi bue forçado
xd talvez falar de dificuldades, cenas assim

TODO TODO TODO

Bibliografia

[Flo,] Flood fill. https://en.wikipedia.org/wiki/Flood_fill.

[Perlin,] Perlin, K. Perlin noise. https://en.wikipedia.org/wiki/Perlin_noise.

[Studios, 2011] Studios, M. (2011). Minecraft. <https://www.minecraft.net>.

[Technologies, 2005] Technologies, U. (2005). Unity. <https://unity.com/>.

[Unity,] Unity. ML-agents. <https://github.com/Unity-Technologies/ml-agents>.