DDA FPS in Unity: Game Development Using Unity

Supervisor: Kemi Ademoye

Programme: MSc Computer Science (AI)

Word Count:

**Abstract**

In this thesis I developed a FPS game developed in Unity, with two Dynamic Difficulty Adjustment strategies, Dynamic Scripting and Reinforcement Learning.

**Table of Contents**

[Introduction 4](#_heading=h.gjdgxs)

[What is Unity? 5](#_heading=h.30j0zll)

[What is DDA? 5](#_heading=h.1fob9te)

[Dynamic Scripting 5](#_heading=h.3znysh7)

[Reinforcement Learning 5](#_heading=h.2et92p0)

[Literature Review 5](#_heading=h.tyjcwt)

[Methodology and Results 6](#_heading=h.3dy6vkm)

[Agile & DevOps 8](#_heading=h.1t3h5sf)

[Source Control 8](#_heading=h.4d34og8)

[Agile Scrum Methodology 8](#_heading=h.2s8eyo1)

[Development Operations & CI/CD 8](#_heading=h.17dp8vu)

1. Project Management
   1. Google Drive
2. Gameplay & mechanics
   1. Shooting & Pickup
   2. Enemy AI Behavior
   3. PCG
   4. DDA
      1. Scoring system
3. Developing in Unity
4. Game Design/Software Design
   1. Good Practices
   2. Software Design Choices
   3. Software Testing
   4. Performance
      1. Multi-threadingx
      2. Memory management
5. Player Testing
6. Conclusion

Appendix

# Introduction

The project involves developing a Unity game with some software design patterns with two main elements, Procedurally Generated Content (PCG) and Dynamic Difficulty Adjustment (DDA).

The theme of the game will be a First Person Shooter (FPS) that is based in a dark dungeon environment. The game consists of two stages. The first stage will be a PCG maze and the second stage will be a predefined map layout which contains the FPS & DDA elements.

The project will be developed using the Agile/Scrum methodology.

Topics

* Developing an efficient and modular Unity game
* Procedurally generating mazes
* Dynamic difficulty adjustment
* Agile/Scrum methodology

# What is Unity?

# What is DDA?

## Dynamic Scripting

Dynamic Scripting (TBF) is a form of reinforcement learning which was implemented on a RPG game, Baldur’s Gate. A rulebase was proposed for each type of character (wizard, fighter, etc.) and each character had a script that was updated in real-time composing rules from its specific rulebase. (TBC) - Each rule contains two values, priority value and weight, which are ranked respectively. The script size for each character type is set to a fixed number of rules.

The two pseudocode demonstrated by Dynamic Scripting, “Script Generation” and “Weight Adjustment”, (is responsible for maintaining the core concept of dynamic scripting). “Script Generation” generates a new script for a character type by using a roulette wheel selection of its rules; rules with higher weights are most likely chosen. “Weight Adjustment” updates the adjustment needed for the weights of its selected rules in the current script by computing the fitness of the agents and its team.

The “Calculate Adjustment” function of “Weight Adjustment” pseudocode includes the primary logic of rewarding or penalizing rules by utilizing the agent’s fitness and a specified break-even value.

Though the Dynamic Scripting has been shown to be successful in various situations (TBF), the strategy had to be adapted to fit into the genre of the game. As opposed to various “rules” which have variable weights, there will be 3 parameters tweaked which are health of enemies, damage of enemies and number of enemies.

## Reinforcement Learning

# Literature Review

# Methodology

There are 3 main components to the game’s dynamic difficulty, Player’s Fitness, Enemy Fitness & Dynamic Difficulty Adjustment. The player’s fitness is compared against the fitness of the enemies and depending on the difference between the two, the difficulty is adjusted dynamically.

The Dynamic Difficulty Adjustment function is defined as follows.

The difference of the player’s fitness and the enemy’s fitness is defined as follows.

In the DDA eq., the threshold was set to be 0.2. With a higher threshold value of

Player Fitness Function

The player’s fitness equation consists of 5 components, (i) number of kills, (ii) actions per minute, (iii) time elapsed, (iv) hit miss ratio and (v) hits taken and is defined as follows.

Enemy Fitness Function

The enemy fitness function is based on two fitness functions, (i) a fitness function of enemies at the current timestep , and (ii) a fitness function of all defeated enemies .

The fitness function is defined as follows.

In Eq. , refers to one live agent, is the total number of alive agents, is the total number of live agents at the current timestep, and along with denotes the starting distance of the live agent from the player and the ending distance of the live agent from the player respectively. The function calculates the distance of the various live agents from the player while considering the current number of enemies against the max possible number of enemies. In this work, was chosen to be 50 based on the assumption of a player with the highest fitness level could theoretically handle. By increasing the value of ??? Intrinsically, the fitness function is a good measure of the current difficulty level of the enemies.

The fitness function is defined as follows.

In Eq. ,denotes the learning rate, refers to a dead agent, is the total number of dead agents and the rest of the notations follows Eq. . The fitness function involves the calculation of all dead agents and the last distance of the agents from the player. The learning rate was set to be 0.1 to minimally influence the fitness of the enemies.

# Agile & DevOps

## Source Control

MDK2023’s utilized both Git and Git LFS during its development cycle. The git repository of the project is currently hosted on the Github [TBF-TODO] platform. The repository requires Git LFS as the model and animation of assets are big files (e.g. animation is ~100MB).

## Agile Scrum Methodology

For the software development of the project, the scrum methodology was adopted. There was a total of 6 sprints throughout the lifecycle development of the project. The various tasks were named as the Product Backlog which is listed in a priority order in the Scrum spreadsheet and written in a “user story” format. Specific “user stories” were selected as the primary focus of respective sprints based on their priority order.

A backlog item in a sprint further contained a priority level (low, medium & high) of the backlog item against other items in a specific sprint, the number of estimated days to develop the backlog item, sprint cycle number, status (In Progress/Done) of and user ownership. This allowed a structured and straightforward collaboration tool for the various members of the team.

## Development Operations & CI/CD

Rapid Deployment (TBI) allows us to both quickly test our new build environments and reliably deploy our game. The Continuous Deployment (CD) tool used, Github Actions, hooks into the Github repository via a config file. The config file, which is stored on Git inside the Github repository, specifies the steps needed to set up the environment, run unit tests, build the Unity game for WebGL and push the built WebGL onto AWS. The test job ensures the unit tests written in Unity runs and passes. The build job ensures the Unity application successfully builds and runs in the specified environment, WebGL.

The workflow of the Continuous Deployment is ran on the latest Ubuntu container and the workflow is defined as follows:

| Workflow No. | Workflow Name | Action Library & Version |
| --- | --- | --- |
| 1 | Checkout repository | actions/checkout@v3 |
| 2 | Create LFS file list | - |
| 3 | Restore LFS cache | actions/cache@v3 |
| 4 | Git LFS pull | - |
| 5 | Cache | actions/cache@v3 |
| 6 | Builds project | game-ci/unity-builder@v2 |
| 7 | Configure AWS credentials | aws-actions/configure-aws-credentials@v2 |
| 8 | Sync output to S3 | - |

Workflow No. 1 ensures that we are checked out in the necessary repository thereby ensuring the container access to all the files of the source code. Workflow No. 2 creates a LFS file list which creates a sorted list of identifiers for all the files tracked by Git LFS and saves it to a file ‘.lfs-assets-id’. Workflow No. 3 restores LFS cache attempts to find the cache associated with the provided key; if found, the cache files are restored to the specified path ‘.git/lfs’. Workflow No. 4 pulls the large files that are referenced in the current commit but not yet downloaded to the local working copy. Additionally, it also resets the HEAD, index and working directory to the state of the last commit to ensure the repository has the latest changes only.

Workflow No. 5 attempts to speed up the build time of the CD workflow by finding the cache corresponding to the ‘key’ and restoring the ‘Library’ directory from the cache. If the ‘key’ is not found, a new ‘key’ is generated and a new cache will be created when the workflow completes. Workflow No. 6 builds the Unity Project to WebGL platform using 3 secret credentials. TALK ABOUT COMPRESSION [TODO]

In order to use Unity in the Github actions to create a build for WebGL, a relevant Unity installation needs to be activated. In order to activate a Unity installation, a valid license file is needed. To generate a license file [<https://game.ci/docs/github/activation>], a relevant activation file has to be generated first which is done through a Github Action workflow. With the newly obtained activation file, it has to be uploaded to license.unity3d.com which returns a download of an .alf license file in XML format. The contents of the file would then be uploaded onto the relevant Github’s repository under the name, UNITY\_LICENSE. In addition to this secret, the two other secrets needed are UNITY\_EMAIL and UNITY\_PASSWORD, with these three secrets, developers are able to use a valid Unity installation to build for WebGL.

Workflow No. 7 sets the three AWS credentials namely, aws-access-key-id, aws-secret-access-key and aws-region which grants programmatic access to the AWS services, specifically AWS S3. Workflow No. 8 pushes the WebGL built files onto S3 and sets the appropriate metadata of various compressed files.

The average time to build and test on Github Actions environment is approximately 10 minutes with caching and 20 minutes without caching.

The Unity application communicates regularly with an internal Web JSON API, which generates a guest token and stores game statistics as persisted information. The internal Web API is developed in Ruby on Rails with a Postgres database hosted on a DigitalOcean Droplet with a Dokku add-on. Dokku

The Unity Web Game communicates with the Backend API regularly through various API calls. Firstly, the Unity Web Game submits a ‘guest token’ API POST request to the Backend API which returns a JSON Web Token. This token gets persisted and stored throughout the game session.

The game statistics stored on Postgres contains the various states of the game when the game’s difficulty is dynamically adjusted and final scores of players which culminates into a global leaderboard.

.secrets file with:

AWS\_ACCESS\_KEY=

AWS\_SECRET\_KEY=

SSH\_DEPLOY\_KEY=

Aws cloudfront - cache invalidation