UNIVERSITY OF YORK

DEPARTMENT OF COMPUTER SCIENCE

ENG1 Group Assessment 2
Team 1

Auber

Impl2.pdf - Implementation

Group Members:

Jonathan Davies

Jamie Hewison

Harry Smith

Zee Thompson

Mark Varnaliy

Documented Code

The code for our Assessment 2 project can be found in the submitted .zip file, within the folder called ‘Code’. We believe that the code provided is sufficiently documented so that another programmer would be able to read over our project and understand what they are working with, and also so that it is clear to examiners what each part of the code meets each part of assessment’s requirements. Our code was created using LibGDX and the IntelliJ IDEA IDE - Our justification for using both of these can be found in the updated Plan1.pdf document.

We have also included the required executable JAR of the game in the submitted .zip file.

Implementation and Significant Changes

In order to create and update the game, we worked through our Architecture plans and Requirements list, ensuring that all functions we’d specified were implemented into our game. The abstract architecture helped us to decide how the Java classes involved in our game should be linked together, from which we were able to produce our code and the subsequent concrete architecture. The following are the major changes we have made for assessment 2:

Operative Abilities (Previously Unimplemented)

Operative special abilities (by which three operatives have unique abilities that make them harder to arrest) were implemented by giving each operative a ‘specialAbilityID’ data tag, as an integer. Most operatives have the specialAbilityID 0, meaning none, while three have unique IDs of 1, 2 and 3, meaning armoured (increased health), increased strength, and increased speed respectively.

Each operative with a special ability also has its own colour (black, orange and blue respectively) to distinguish them from the standard (grey). They also have unique antennae on their characters so the player is not entirely reliant on colour, helping to fulfil **NFR_PLAYABILITY_ACCESSIBILITY**.

This addition completes the functional requirement **FR_OPER_ABILITY** with only minor changes to the coding of the operative actors.

Demo Mode (Previously Unimplemented)

The demo mode runs the game as it is but with the Player actor replaced with a PlayerDemo actor.

This replacement uses the same path finding algorithm the operatives use however the target is any remaining operatives in the game. All user input and saving functions are disabled for this mode. The demo player will find and attempt to capture all operatives before the end of the game.

The demo player will not actively seek out powerups and for this to be fully implemented the demo player will need to pick up any power ups it finds.

This addition partially completes the functional requirement **FR_GAME_DEMO** with little change to the original architecture. The only change being the replacement of a user based input with the operative path finding algorithm**.**

Power Ups (New Requirement)

Power Ups were implemented by creating a new class. Each powerup type is contained within the PowerUp class which is an extension to the LibGDX Actor class.

Powerups are obtained when a user attacks them. The powerups use the already existing hit collision system implemented by the previous team and allows the powerups to interact with the player through that system.

One large change implemented is frame delta time animation calculations rather than frame count based animations. This means the animations will run at the same speed independent of the framerate.

Powerups are placed around the map as well as dropped from captured operatives. This addition completes the new functional requirement **FR_PLAYER_POWERUP** and with little change to the original architecture**.**

Difficulty Setting (New Requirement)

Difficulty settings were implemented by creating a new DifficultyScreen class.

This screen gives the user 4 options Easy, Normal, Hard, Demo. The difficulty is then passed to the GameScreen class constructor. Difficulty is implemented by increasing the operatives health, and damage. The player's speed is increased on Easy. The players reliance on power ups and healing is greater the higher the difficulty is making the game challenging and exciting.

This does not change the original game architecture but does change the menu screen format. This new addition fulfils the requirement **FR_GAME_DIFFICULTY**.

Game Saving (New Requirement)

Game saving was implemented via LibGDX's built in ability to save game preferences, which can be manipulated to store game save data.

On saving, the game writes all the relevant information about Auber to the preferences file in its original data type, such as their current coordinates, health, power up abilities, etc. It also saves serialised arrays containing information on the remaining systems, operatives and power ups on the map. Saving as serialised arrays makes this data easier to parse on resuming. When loading a saved game, Auber's default information is overridden with the saved details, making them exactly the same as when the game was saved. The code which places the systems,

operatives and power ups on the map also deserialises the saved data and loads from this, rather than the original information in the game data file.

This completes the functional requirement **FR_GAME_SAVE** with the only major changes to the code being how all the actors are placed on the map, since the default data is overwritten. No other changes are necessary, as the game runs as normal once loaded.

Other:

Some changes were not significant enough to go into detail about since they barely change the implementation, but are still worth considering:

- Game pausing has been implemented. This is done simply by setting a boolean to True when the game is paused. All active features of the game check this before making any action, if it is true, then the action does not continue. This does not link to an explicit function requirement, but was implemented since it is a standard in computer games.
- A mute mode was implemented to fulfil **FR_GAME_MUTE**. This adds a button to the home menu that will mute all music and sound effects of the game, by setting a boolean to True to show it is muted. All sounds in the game check this before producing noise.
- Slight changes were made to fulfil **FR_UI_SCALABLE.** The game is now aware of the size of screen it is running on, and will scale all images, buttons, text and characters to match, so the image does not look distorted.

Features that are not (fully) implemented:

We believe that we have met all of our goals for this project, and that all requirements have been implemented to a suitable degree. There are a few requirements that have been met, but could still be improved if we had to take the game further, these include:

- FR_GAME_DEMO: The AI for Demo mode will not actively seek out power ups, but can use them if picked up along the way. The demo mode is still fully functional.
- FR_UI_SCALABLE: There is no option for the user to change their resolution manually, but the game will automatically scale to full screen on any display, which meets the requirement.
- NFR_PLAYABILITY_ACCESSIBILITY: While there have been no accessibility modes explicitly implemented, the game is able to be played by someone who is colourblind, etc. Each of the operatives have a different antenna, and the game is fully playable as tested by applying different colour filters to our screens. The game is still fully playable, and as such the requirement is sufficiently fulfilled.