Module: ENG1/ASSESSMENT1

Title: Implementation

Charles Stubbs
Annabelle Partis
Kieran Ashton
Yu Li
George Tassou
Alex Shore

Part A

https://github.com/stubbs774/runtimeerrors-two/tree/main/game

Part B: Implementation, Architecture and Requirements

Note on Format and Completeness

Our code is a direct implementation of our architecture. The relationships between entities in the code are the same in the architecture diagrams. Every change made to the architecture and code was necessitated by a requirement. These are listed below, as are the classes in the code and architecture affected by these changes. All requirements stated in the requirements document are fulfilled. However, the stakeholders requested in assessment one that the game have a demo mode, but the previous developers omitted this from their requirements and did not implement it. Due to the close deadline of this assessment, we have prioritized fulfilling our high-priority requirements over correcting this.We have often used getter and setter methods in the code to avoid directly accessing variables of other classes.

The updated architecture can be found here:
https://raw.githubusercontent.com/stubbs774/runtimeerrors-two/main/website/extradocs/Req2.pdf
The updated requirements can be found here:
https://raw.githubusercontent.com/stubbs774/runtimeerrors-two/main/website/extradocs/Arch2.pdf

The classes mentioned in this report can be found at these links:

Infiltrator	https://github.com/stubbs774/runtimeerrors-two/blob/main/game/core/src/com/team5/game/Sprites/Infiltrator.java
InfiltratorAI Behaviour	https://github.com/stubbs774/runtimeerrors-two/blob/main/game/core/src/com/team5/game/Sprites/Pathfinding/InfiltratorAIBehaviour.java
NPCAIBehaviour	https://github.com/stubbs774/runtimeerrors-two/blob/main/game/core/src/com/team5/game/Sprites/Pathfinding/NPCAIBehaviour.java
Player	https://github.com/stubbs774/runtimeerrors-two/blob/main/game/core/src/com/team5/game/Sprites/Player.java
Trap	https://github.com/stubbs774/runtimeerrors-two/blob/main/game/core/src/com/team5/game/Sprites/Pathfinding/Trap.java
GameController	https://github.com/stubbs774/runtimeerrors-two/blob/main/game/core/src/com/team5/game/Tools/GameController.java
Teleporters	https://github.com/stubbs774/runtimeerrors-two/blob/main/game/core/src/com/team5/game/Sprites/Teleporters.java
MainGame	https://github.com/stubbs774/runtimeerrors-two/blob/main/game/core/src/com/team5/game/MainGame.java
LevelScreen	https://github.com/stubbs774/runtimeerrors-two/blob/main/game/core/src/com/team5/game/Screens/LevelScreen.java
MainMenu Screen	https://github.com/stubbs774/runtimeerrors-two/blob/main/game/core/src/com/team5/game/Screens/MainMenuScreen.java
GameController	https://github.com/stubbs774/runtimeerrors-two/blob/main/game/core/src/com/team5/game/Tools/GameController.java
Constants	https://github.com/stubbs774/runtimeerrors-two/blob/main/game/core/src/com/team5/game/Tools/Constants.java

NPC	https://github.com/stubbs774/runtimeerrors-two/blob/main/game/core/src/com/team5/game/Sprites/NPC.java
PauseMenu	https://github.com/stubbs774/runtimeerrors-two/blob/main/game/core/src/com/team5/game/UI/PauseMenu.java
PlayScreen	https://github.com/stubbs774/runtimeerrors-two/blob/main/game/core/src/com/team5/game/Screens/PlayScreen.java
LoadScreen	https://github.com/stubbs774/runtimeerrors-two/blob/main/game/core/src/com/team5/game/Screens/LoadScreen.java

Changes to fulfill UR_INFILTRATOR_ABILITIES:

Classes modified: Infiltrator, InfiltratorAIBehaviour, NPCAIBehaviour

Justification: This was a significant part of the first set of requirements specified by the stakeholders, and without it the gameplay would lack variety. It also helps to fulfill requirement UR_UX which requires that the system should offer a pleasant user experience.

The previous development team did not fulfill the requirement UR_INFILTRATORS_ABILITY, or translate it into a functional requirement. In their implementation, they provided one infiltrator ability in the form of a function changeSkin(), located in the infiltrator class shown clearly in the architecture diagram. We therefore had to implement two more infiltrator abilities, though first we added an FR_INFILTRATORS_ABILITY to the requirements to guide our development.

We created two abilities, sprint and quick sabotage, both of which are activated through calling activateAbility(int index,int probability) found inside class InfiltratorAIBehaviour. Two data structures were added to this class to keep track of these abilities: abilities available, an array of booleans with each boolean referring to whether the respective ability had been used, and quicksabotage, a boolean variable that the code uses to determine whether the infiltrator will use the quick sabotage ability when they reach the next system.The infiltrators also uses the NPCAIBehaviour class, where their speeds are set. In this class, a boolean variable sprint, and a float variable sprintTimer were created to modify the infiltrator's speed while the ability is active. An addition was made to the move() function in the same class to update the sprint timer every frame.

The classes Infiltrator, InfiltratorAIBehaviour, and NPCAIBehaviour were added by the previous developer team. Infiltrator is a subclass of NPC, and InfiltratorAIBehaviour is a subclass of NPCAIBehaviour, as specified in the architecture diagrams. None of the methods we coded directly access variables inside another class; if they need to modify or access any, they call getter and setter functions from that class instead.

Changes to fulfil FR_AUBER_ABILITIES:

Classes modified: Player, Trap, GameController, InfiltratorAIBehaviour, Teleporters

Justification: This was requested by the stakeholders mid-development, with a high priority, and therefore prioritised, despite the extended time required to conceptualize and implement the unique abilities. This also helps to fulfill the requirement UR_UX which requires the system offer a "pleasant user experience".

FR_AUBER_ABILITIES requires the implementation of five different abilities that can be activated by the user. The abilities we chose were system invulnerability, which prevents any imposters from sabotaging the systems for a duration, sprint, which allows the user to traverse the map faster, wrist teleport, which allows the user to teleport once per game without requiring the user to find and click on a teleporter, slow infiltrators, which force infiltrators nearby the player to slow down even if they are using their sprint ability, and finally motion sensor, which allows the player to place a motion sensor on the map which will change texture to indicate an infiltrator is within its bounds. All of these abilities can only be used once, and each are activated by one of the number keys in the range of one to five.

Four data structures were added to the Player class to track the abilities: two arrays of booleans, abilityAvailable and abilityCurrentlyActive, where the abilityCurrentlyActive is a pointer for abilityAvailable. The values in abilityAvailable start off true, and are negated by function activateAbility when the user presses the allocated key. abilityAvailable also sets the value relating to each ability in abilityCurrentlyActive to true while the ability is active. The duration of the ability is given by the integer variable abilityTimeLeft, which is decremented by update(). The index of the current ability within the boolean arrays is given by the integer currentAbility, which will be set to 100 by update() if the abilityTimeLeft counter reaches zero.

In order to facilitate sprint ability, integer variables sprintSpeed and normalSpeed were added. When the sprint key is pressed, the speed variable is assigned the value of sprintSpeed by checkInputs() in the Player class. When the abilityTimeLeft counter reaches zero, speed is set to the value of normalSpeed by update() in the Player class.

Before running the code in InfiltratorAIBehaviour to break a system, the game first accesses the abilityCurrentlyActive array and uses an if statement to check if system invulnerability is active. If so, the code that breaks the system is skipped.

A subroutine, checkTeleports, was added to the Teleporters class that opened the teleport map if the wrist teleport ability was active. This was called in the GameController class. Once the player has teleported, the next line of code in GameController calls the function deactiveTeleportAbility() from the player class to set the variables to prevent the ability being used again.

The function update() in the InfiltratorAIBehaviour class calls a function in the same class named slowingAbility() that checks if the value corresponding to infiltrator slowdown is true and if the infiltator's position coordinates are close to that of the player, and if so, calls a function from its superclass, decreaseInfiltratorSpeed, to set the value of the boolean slowed to true. The move() function in this class has an if statement to check if slowed is true, and if so, returns the reduced speed. slowingAbility() detects if the ability is no longer active and calls super.increaseInfiltratorSpeed, which changes the slowed variable in NPCAIBehaviour and causes move() to return the original speed again.

The last ability is the motion sensor, or motion trap, and the Trap class was created for it. It is a child of the Libgdx object Rectangle, and stores three variables: default skin, which contains the texture of the unactivated motion trap, and alert skin, which contains the texture of the activated motion trap, and alerted, which stores a boolean value indicating whether an infiltrator has been detected standing in the bounds of the Rectangle object. The class has four functions, the first of which is the constructor. This instantiates the superclass and sets the position of the motion trap on the map. The function getSkin() is a getter method that returns alertSkin if alerted is true, else returns defaultSkin. Meanwhile functions alert() and resetAlert() are setter methods that are called to change the value of alerted.

Two variables were added to gameController to facilitate the motionTrap ability: the first, the boolean variable motionTrapExists, which indicates that the motion trap is currently on the map. If so, draw(SpriteBatch batch) runs the code to call the function getSkin() from Trap and draws on the screen the texture returned. The other variable is motionTrap. It contains the instantiated Trap. Two functions were added to gameController. These are updateTrapAlert and updateTrapState(Player player). updateTrapAlert iterates through the array of infiltrators and passes their location coordinates into the contains(float x, float y) function provided for Trap by Rectangle. If this function returns true, then updateTrapAlert calls the alert() function from Trap. Else it calls resetAlert().

A function shouldCreateMotionTrap(boolean motionTrapExists) was added to Player. It is called from gameController. An if statement will check the relevant value in the abilityCurrentlyActive array, and the state of motionTrapExists. If the value in abilityCurrentlyActive is true but not motionTrapExists, this indicates that the ability is active, but no motionTrap object has been created yet, so it returns true, prompting the gameController class to instantiate Trap. Else it returns false.

Changes to fulfil FR_DIFFICULTY

Classes modified: MainGame, LevelScreen, MainMenuScreen, GameController

Justification: This was an important change to the game requested by the stakeholders, to provide gameplay variety that would be otherwise lacking. We also judged it easily implementable.

The class LevelScreen was to handle difficulty options. It has similar variables to MainMenuScreen, with the addition of ImageButton objects easyButton, mediumButton, and hardButton, which are assigned the corresponding textures in the function setupButtons(). Inside this function there are a number of nested functions, all called clicked(InputEvent event, float x, float y) each linked to a button. When the button is pressed they call a setter method from MainGame, setLevel(int level) that changes difficulty level. MainGame also has a function getLevel() which returns the selected difficulty level.
GameController was modified to call getLevel from MainGame. It checks the difficulty level using an if statement, and sets the noNPCs and noInfiltrator variables to different values depending on the difficulty. On the hardest difficulty, there are many more npcs and two less infiltrators than in the other modes, which makes them much harder to identify amongst the crowds.

The changes made to MainMenuScreen were minor: we added an ImageButton variable called levelButton to house the texture and provide button functionality. A clicked(InputEvent event, float x, float y) function was added inside setupButtons(), linked to levelButton, to instantiate LevelScreen.

Changes to fulfil FR_MUTEABLE_SOUND

Classes modified: MainMenuScreen and Constants, with minor changes to all classes that implement the Screen interface as well as InfiltratorAIBehaviour, Infiltrator, NPC and PauseMenu.

Justification: This requirement was provided with the first brief and was essential as the product would be used in teaching environments that require quiet. However, the previous developer team did not provide it despite having audio in their game.

The variable volumeMultiplier was added to the static class Constants. We decided to put it in this class because while its value could be changed from the main menu, it would remain constant throughout gameplay. We added an ImageButton variable called muteButton into the MainMenuScreen class to control the volume. In the setupButtons() method of this class we created an array of Image objects, called muteButtons, to contain all the textures: one to indicate sound, one to indicate silence, and variants of these buttons in selected mode. We also created an integer variable, currentMuteButton to store the index of the current texture of the mute button in the array. We also created a nested function linked to muteButton, clicked(InputEvent event,float x,float y), to change the value of currentMuteButton when clicked and change the value of volumeMultiplier between 1 and 0. We then changed all the audio cues in the game so that the volume parameter would be multiplied by the volumeMultiplier.

Changes to fulfil FR_SAVE_LOAD

Classes modified: PlayScreen, LoadScreen, PauseMenu, GameController, SystemChecker

Justification:This was requested with high priority by the stakeholders and makes the game more accessible by allowing players to easily take a break, thus was worth the implementation time.

Three variables were added to the class PauseMenu: saveButton of the type ImageButton, saveOffset of the type Vector2, responsible for the positioning of the save button, and Preferences, of type prefs. The latter, when instantiated, uses a libgdx function to create a file to save data structures. The saveButton variable is linked to a clicked(InputEvent event,float x,float y) function that calls putInteger(String key, int data) and putFloat(String key, float data) from the prefs object. These functions use the file contained in prefs to create a dictionary with the variable in.

LoadScreen was created to house loading options. It implements the screen interface, and variables and functions similar to those inMainMenuScreen, including a setupButtons() function. Both the loadButton and newButton instantiate PlayScreen, but they also pass a boolean variable loadGame, which is true if loadButton is selected, and false otherwise.

We also added a loadGame variable to PlayScreen, storing the boolean received from loadGame and then passing it to GameController.

GameController also has a pref variable (again of type Preferences), which links to the dictionary file. It also has variables for the integers and floats that were saved. If GameController receives a true value of loadGame, its constructor will begin calling setters from different classes and loading them from the dictionary to reassemble the state the game was saved in.