Impl2

Enigma / Team 29

Michael Beal
Lucas Solomon
Thomas Auburn
Ryan Early
Inna Strogonova
Daniel Pearce

Link to github containing the repository of our code: https://github.com/Dauthlin/Auber2

Implementation

Our code implements the architecture displayed in the modified Architecture deliverable as we tried to implement our code in the same style and layout as the previous group meaning that our changes would fall under the same type of architecture that was used before and would aline with our changes that we made. The main changes that we made did not require new classes to be created but fell into the old classes that were already implemented which meant that the underlying architecture of how the code functioned was not altered by our additions. In the cases where new classes where required such as the classes to create the new abilities where stored in the same places as the other classes of this type allowing the group of classes to be collected together.

To accomplish the new user requirements set out in the 2nd half of the project we had to incorporate a number of new features into the game. We needed to be able to save the game, as shown in our new state diagram, and to do this we added a feature to be able to access an in-game menu to select the option to save or load a previous version of the game, this was carried out by a algorithm which switched to the pause screen and prevented the game from disposing of its data when the screen was changed so the game could be reloaded. We also had to create the option to change the difficulty of the game and we did this by creating a new screen right after the player presses ‘play’. The new difficulty screen, allows the player to select a difficulty from the options: ’easy’, ‘medium’ or ‘hard’ and once one of them is chosen, the game will start. This information would then be passed down to the game when generating the game. To add this menu we simply redirect the play button on the main menu to this screen, rather than directly to the game. We next added the feature to actually save and load the game, this was accomplished by using json to store the positions of all entities and the values of specific variables relating to the current state of the game and saving them as a LIBGDX preference.

The last of the new user requirements is to implement five special power-ups that give Auber abilities to assist the user to win the game. The first challenge was to implement the random creation of the powerup sprites on the map. To accomplish this, we first created two new classes which act in a similar way to the classes required for the creation of the infiltrators. The first class is the class for the powerups themselves and their methods, the second class is used to generate all the powerups and store each of these powerups in an array.

The powerups we chose to implement are invisibility, speed boost, invulnerability, increased arrest range and preventing the infiltrators from moving. These were implemented through changing the attributes of Auber itself, or through changing the way in which other classes interacted with Auber. After implementing the powerups, as with the other entities we created two classes for saving, one for modelling each powerup and another for collecting the data of all the powerup models for saving.

systematic report

Requirement ID	Class Name	Change	Justification
FR_PAUSE_SCREEN	PauseScreen	Added a pause screen to the game.This uses a modified version of the code made to create the main menu with different names and functions added to the buttons.	This change was necessary so that the game could be saved or loaded at any point during the game's run time. This allows the user to pause the game at any point and save it without losing progress as the game pauses. To achieve this we had to disable the automatic disposal of game data whenever the game switched screens so the game could resume. A minor side effect of this is that a function must be called whenever the data must be deleted instead of this happening automatically.
UR_SAVE_LOAD	CrewModel, PlayerInfo, PrisonerModel, PowerupModel	Added new classes to store the different information about each section of the game that must be saved.	New classes were required as there were no current classes that were appropriate for this task. Each new type of data to be stored needed its own class so Json could serizal the classes separately. Players are a special case and do not need a model to represent them as they are only initialized once.
UR_SAVE_LOAD	INFInfo, PrisonerInfo, PowerupInfo	Added a new class to create instances of the data storing classes and then use json to serialize and save them in the libgdx preferences feature.	Now we have classes which can store the information about each data point. We must initialize them and save them. This needs to be done outside of these classes so that they can be saved as an array of objects containing the relevant data. Json can then be used to save them.
UR_SAVE_LOAD	LoadingGame	Set the game when created to either be in a loading state or a new game state	This means that when loading into a previous version of the game, the game will know that you are loading a past save and will know that it needs to set relevant information to be the same as the save files information instead of loading a new default game.

Requirement ID	Class Name	Change	Justification
UR_SAVE_LOAD	PlayScreen	Changed how infiltrators and crew were initialized	As you must be able to load into a previous save of the game the infiltrators and crew's position and goals must be set to be the same as the game state when the game was saved. This means they must be set when they are created
UR_SAVE_LOAD	Infiltrator, CrewMembers	Changed how the data in crew and infiltrators was stored.	Previously this data was not easily accessible and was required to save the game. We needed to make the data in a serializable data type as it was originally using libgdx arrays as well as make the data accessible in a useful way.
UR_SAVE_LOAD	Infiltrator, CrewMembers	Changed how sprite information is stored.	As sprites can not be serialized the whole system for NPCs storing sprites had to be changed. Their sprite is selected randomly when initialized and this random value is stored along with the sprite itself, this means that when the NPC is saved the value the sprite is created from can be saved rather than the sprite itself and can be reinitialized from this value.
UR_DIFFICULTY	DifficultyScreen	Added a difficulty selection screen when creating a new game.	This screen is needed to select the required difficulty. This screen is located past the main menu when you launch the game. It changes the amount of crew and insulators that spawn and the amount of incorrect guesses you can make.
UR_SAVE_LOAD	SavingGame	Created a main saving class which collected data from the other classes which collected specific data.	We created a class which would serialize all the other classes which collected data about what needed to be saved using Json. This meant that all the saving data was in a single Json file which could then be saved.

Requirement ID	Class Name	Change	Justification
UR_SAVE_LOAD	MainMenuScreen	Added loading button to the main menu	Players may want to be able to load directly into their previous saves rather than having to load a new game then from there go to the pause screen and load their old saved game.
UR_PLAYER_ABILITIES	GeneratePowerups PlayScreen	Created a class for generating instances of the powerup class and storing the powerups in an array	This new class creates a powerup instance and then adds it to an array, as well as functionality for removing a powerup from the array once it has been picked up by the player. We altered the PlayScreen class to incorporate this and run the function that initialises the powerups. This array is referenced when saving the game as it contains all active powerup instances.
UR_PLAYER_ABILITIES	Powerup	Created a class for the powerups	The powerup class is the main class for the powerups that the player can collect. Each new instance is created by the GeneratePowerups class and gives the powerup and ability and its associated sprite. The class contains a method to check the distance of the powerup instance from the player and if it is close enough, sets the player's active ability to the powerup's ability and removes the powerup.
UR_PLAYER_ABILITIES	Player Infiltrator	Added active ability property and five player abilities	First we had to add a property of the player to store its active ability. This variable will determine which ability is being used. A ten second timer for each ability was also added so that the active ability is set to 'None' after the ten seconds and the effects are temporary. Three of the player's abilities are in the Player class, one doubling the player's speed, another preventing the player taking damage, and another increasing the player's arrest radius. The last two powerups affect the infiltrator class, making the player 'invisible' (making the infiltrators unaware of the player's presence) and immobilising infiltrators.