CS 319 - Object-Oriented Software Engineering

Project Final Report

Animal Uprising

<u>Group 3-A</u>

Bora Ecer

Ata Gün Öğün

Albjon Gjuzi

Tanay Akgül

16.12.2017

1. Introduction:

The final report includes implementation process, user guide, deviation from Design Report, and conclusion of the whole project from when it started up to the last day. Luckily we do not have anything left to implement. In the implementation process the work we have done will be explained. In the user guide the system will be explained briefly parallel to the Analysis and Design Report. We will provide some screenshots that guide the user on how to play the game in different parts of it. In the deviation from Design Report, the changes from the Design Report of the project will be explained. Basically we will explain what we hadn't thought of in the Design Report that was included in the system. There are some classes that we did not have there and their functions will be shown briefly in this report.

2. Implementation:

The implementation of the program started right immediately after the Design Report, because we had an overall idea of the classes that were needed and the relation between them. The most difficult part was trying to understand and learn how the 2D graphics work in Java. We had a project that had many objects which was good for the purpose of the course, but difficult for the life of implementers, because as every Object Oriented system, this one had his issues too.

Basically our implementation of GUI resembles the MVC pattern discussed in the classroom. This can be understood from the class relations that we have in our Design Report. The Model, View and Controller are divided in subsystems there.

The Model subsystem is based on the Game Model subsystem that consists of GameObject class which is then extended by every other object that our program will have such as Hero, Allies, Enemies or Castle.

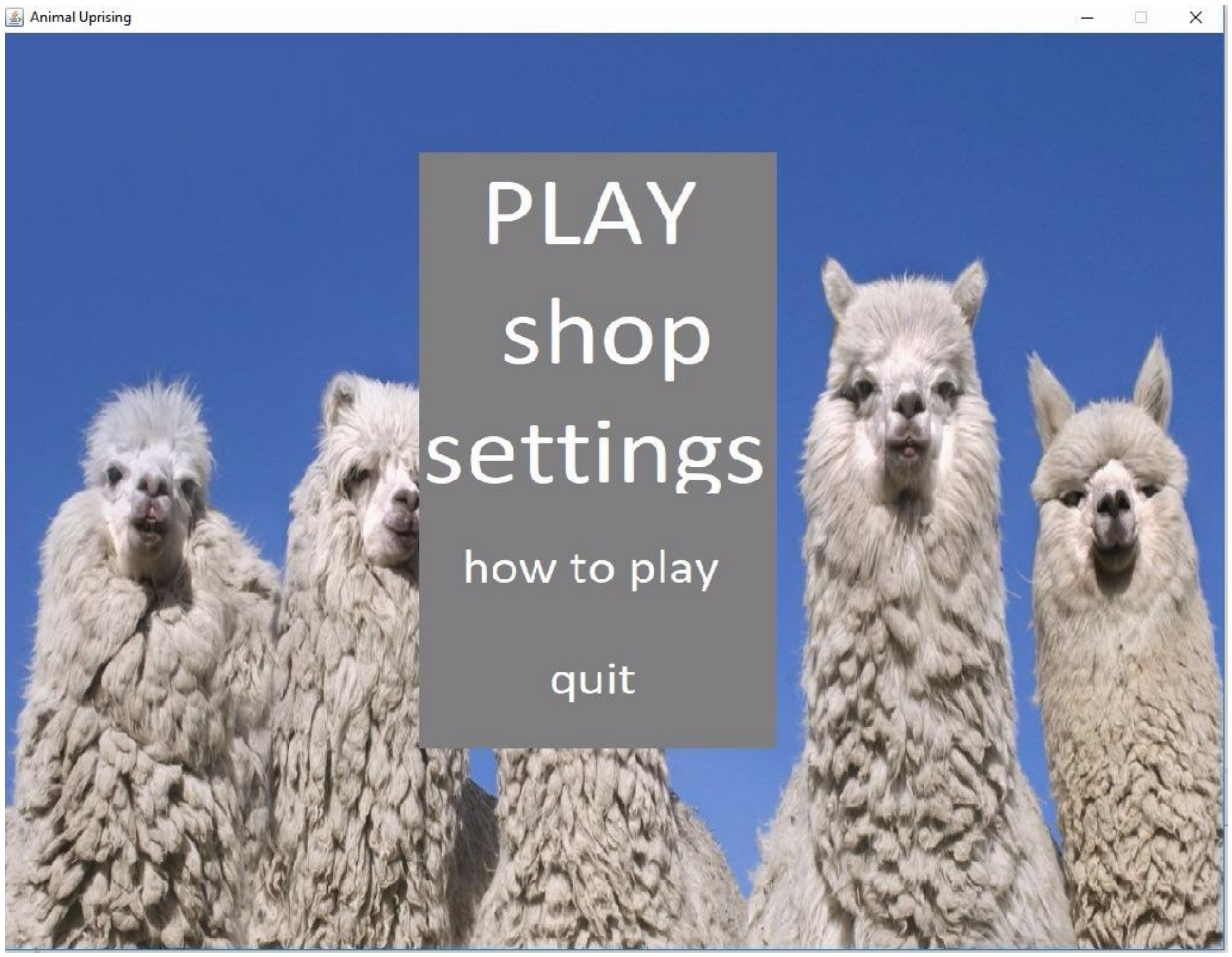
The View will be based on the User Interface Management subsystem with main class the GameEngine class which will create the main frame in which the game will be played.

These two main components if we can call them like that, will then be connected with the Game Control subsystem which starts if we can say so with the GameManager class, and as the name suggests this class will manage everything that happens in our application. The first will be used from ObjectManager class in the Control subsystem to keep track of all objects in the game. The later will be an instance object inside the GameManager class.

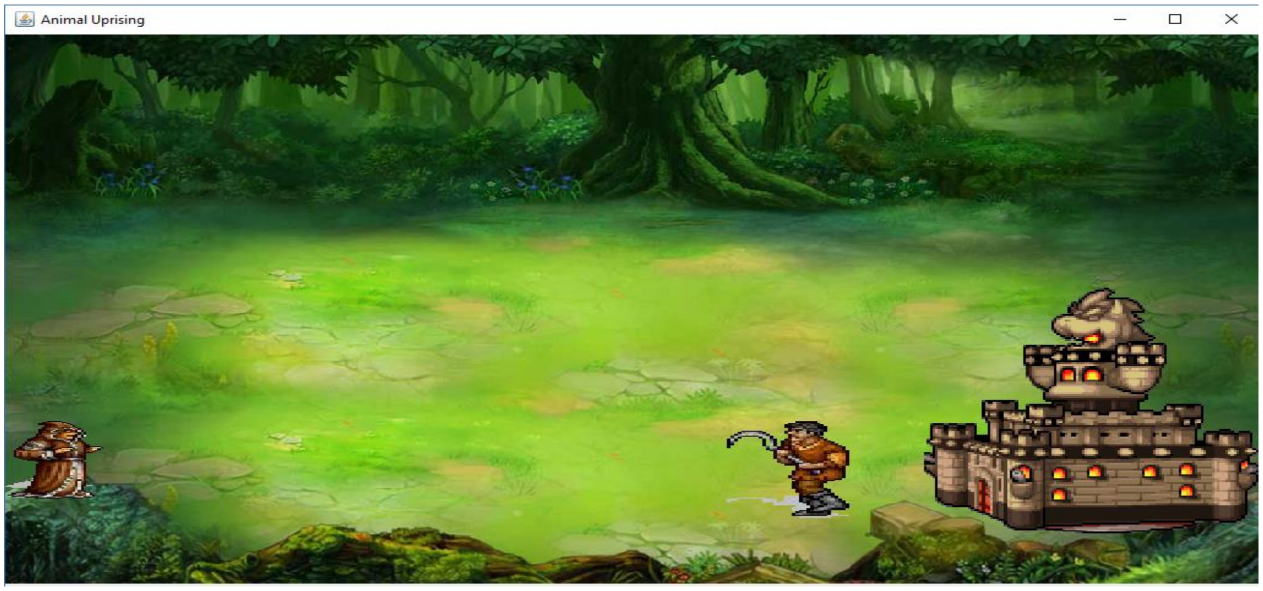
We do not think it is necessary to explain these subsystems again as we have done them in the Design Report, so we thought to explain only the new ones that came out after we submitted the report.

3. User Guide

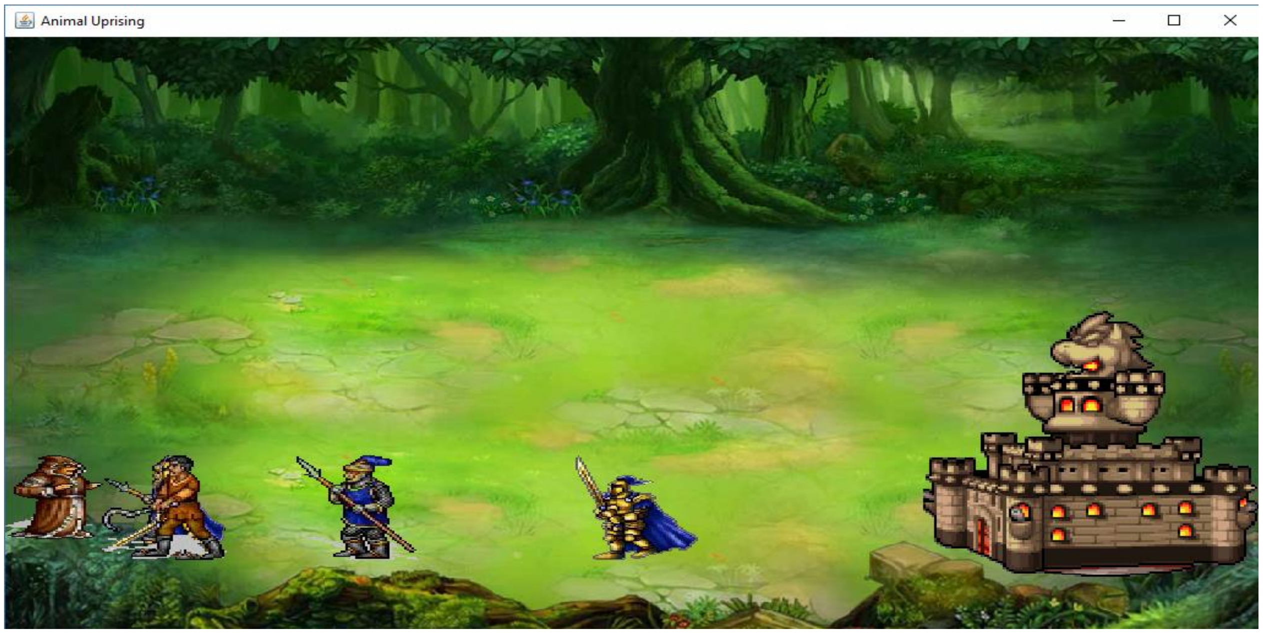
First of all, the game needs some installation process. We have used Eclipse IDE during the development stage and Eclipse does forbid undocumented API as default and it sets the default settings, every time the project is imported, and in our project, we use sun.audio which is not part of Java API. So the user should do the following process if he/she is using EclipseIDE.
Window-->preferences-->Java-->Compiler-->Errors/Warning-->Configure Project Specific Settings--->Java-->Compiler-->Errors/Warning-->Deprecated and restricted API ---> Forbidden reference (access rules): --> ignore



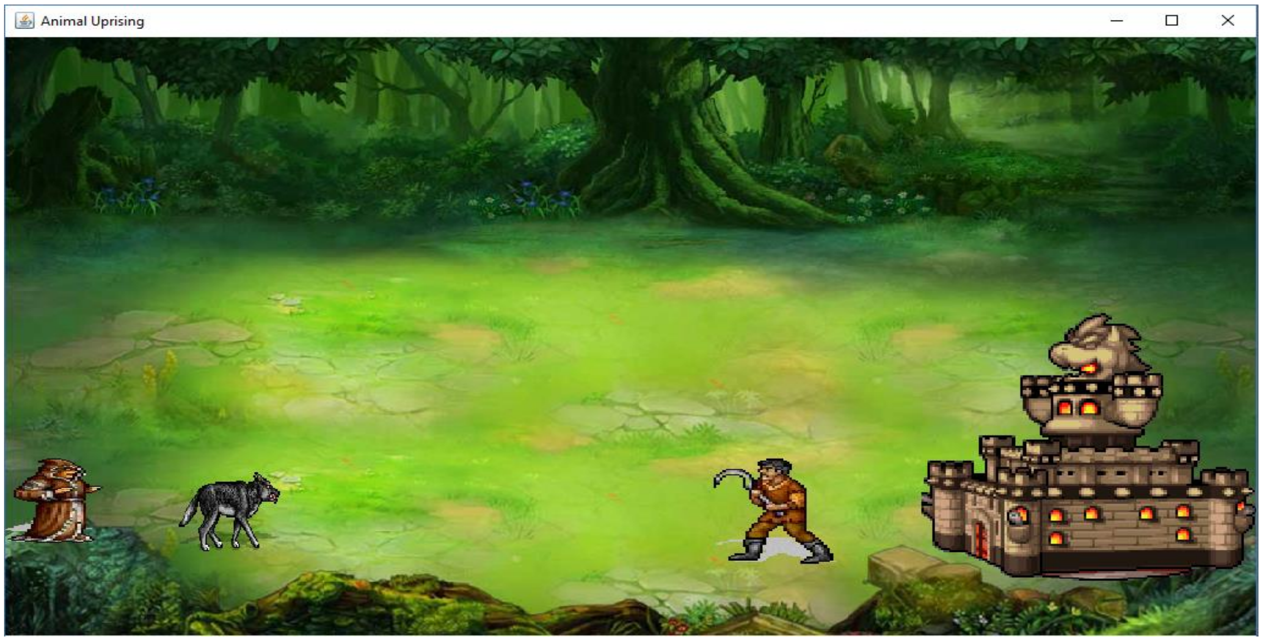
Animal Uprising is a strategy/adventure game similar to those ancient castle games which summon soldiers to protect their own castle and attack the nearby castles to win territories. The difference in our game is that the player will be able to control a hero rather than having a castle. During the game, in order to destroy the enemy castle, the player will have to carve a path to the enemy castle by defeating the enemy soldiers. The player can use the keyboard buttons A and D to move left and right respectively. The player cannot move vertically because the point is to go and attack the castle which will be in the same y-position with the hero.



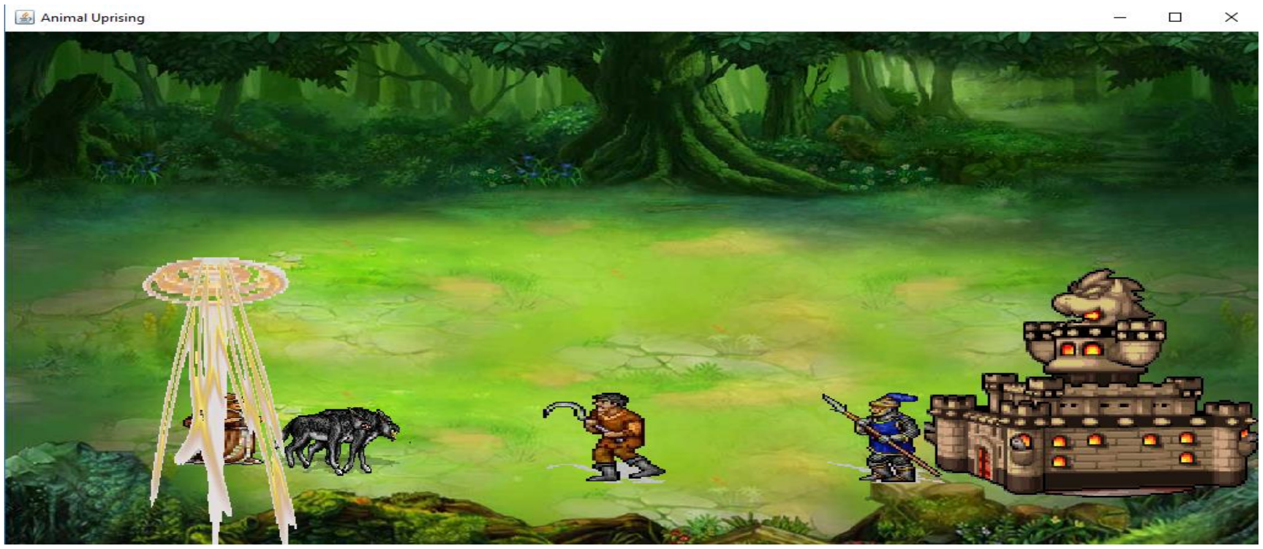
Our player cannot attack the castle itself because the enemy castle summons its soldiers to defend itself.



The hero can summon ally soldiers in order to have a bigger striking force towards enemy soldiers. The summoning of the allies can be made from the numbers 1 - 4 in the keyboard. We chose keyboard numbers because it is easier and more accessible than other keyboard buttons.

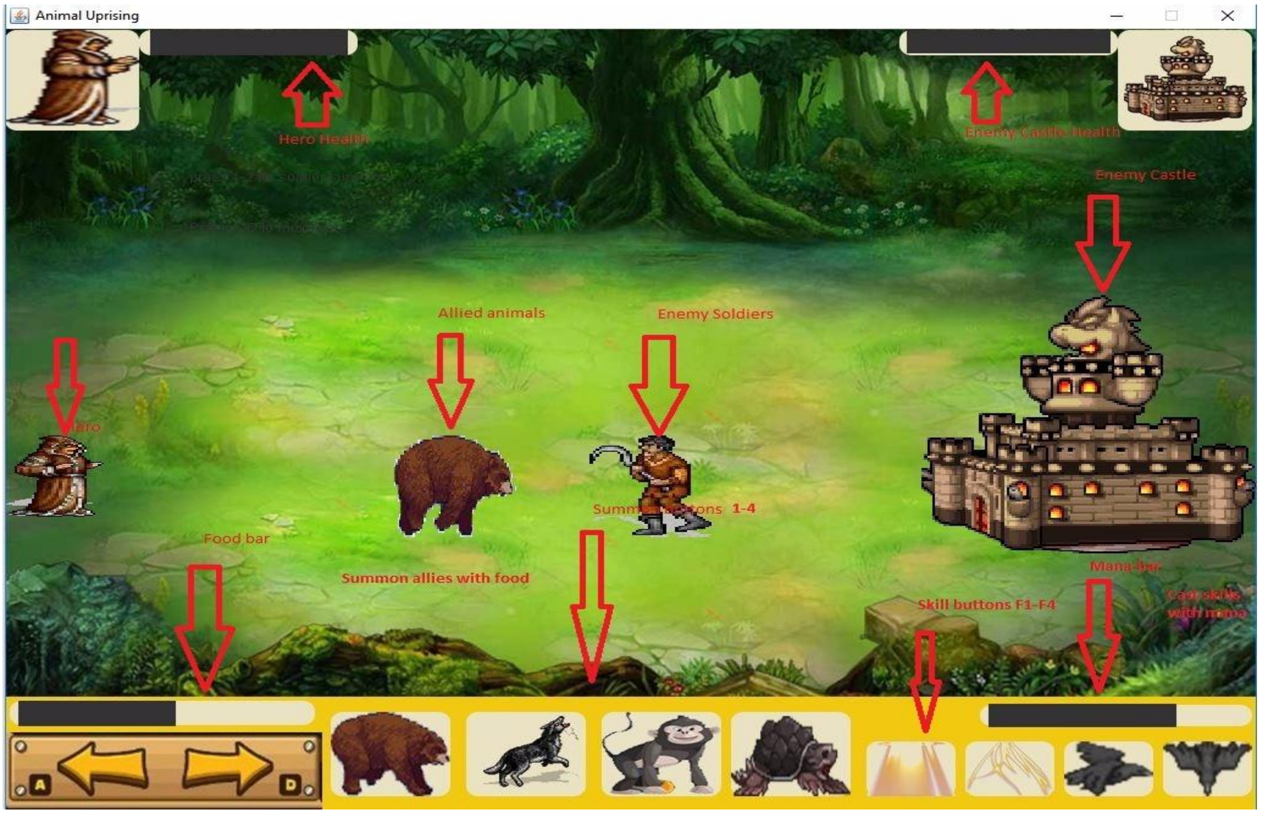


Meanwhile the skills of the hero can be accessed form buttons f1 – f4. The reason for choosing these buttons is the same as the buttons for summoning the allies.



When the user strikes, he will have to wait for a certain amount of time until he can unleash another attack on the enemy soldier. The same thing is valuable for summoning the ally. If one ally is called, the player has to wait for a certain amount of time called cool down time until he can summon another ally. Cool down time is applied to both, skills and summoning.

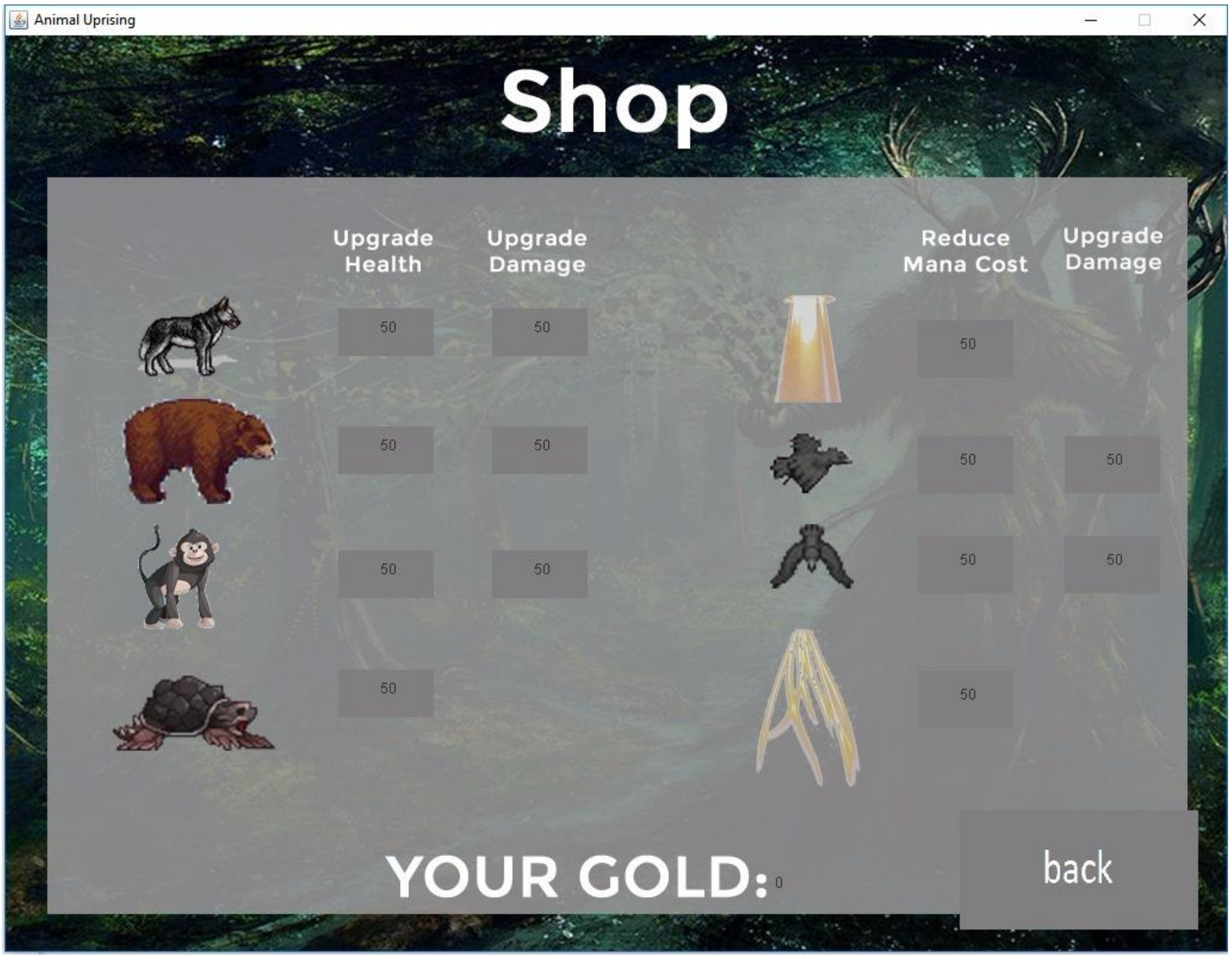
Once the player and allied soldiers reach to the enemy castle, they can attack it to tear it down.



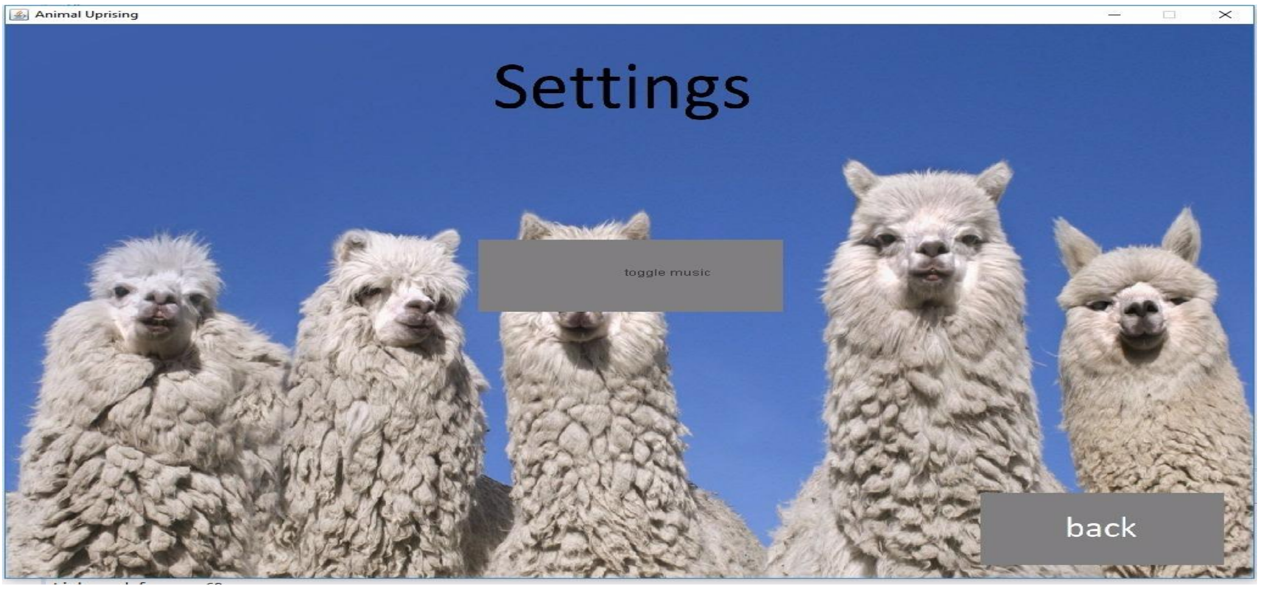
The player hero will be able to cast both defensive and offensive skills which either help the allies or harm the enemies. Because of enemy soldiers attacking all the time our hero has to make the right choices in order to be able to defeat the castle, otherwise his allies will die and he will suffer the force of the enemies and will eventually die.

Upon finishing each level and destroying all enemy soldiers the player will gain coins which will allow him to upgrade skills or soldiers. The player will be able to pause the game and continue whenever they want to.

The last but not the least important feature of our system is the shop menu, in which the user can buy upgrades for the skills of the hero or for the allies. Also, the player will be able to replay completed levels to gain more coins. If the player is not feeling comfortable with how the game is going up to a certain moment he can pause the game and restart the level.



User can also toggle music on/off in the settings menu.



4. Deviation from Design Report

As every other project in the world also ours had its changes from the Design Report too. The implementation phase made us create some other classes inside the subsystems we had in order to have a better interface and connection between the classes. Luckily no major change had to be done. All the changes were minor and they were mostly related to the things that we didn't implement in the first iteration.

Since our game has three main subsystems I am going to explain the deviations for each of them separately, because of the functionality and purpose of each of them.

4.1 Changes in Control Subsystem

a) **CollisionManager class added.**
Being amateurs we had made the mistake of not thinking which class would take care of the attacks and collisions that would occur inside the game, so we had the idea of creating a class specific to this purpose and it came out to be a really good idea.This class gets a CollisionBox from each GameObject and checks whether they intersect with each other or not. For example if a dog ally crashes onto an enemy soldier his health points should be deducted according to the attack power of the enemy.

b) **LevelManager class added.**
Since our aim when we wrote the Design Report was to first have a functional level we did not pay attention to the other level, since we knew it would be easy to update the levels once you have the first one. Basically, this class simply increases the health of every object in the game by a fixed amount of 50 health points. It also increases the number of the unlocked levels so the user does not have why to start from the beginning the next time he plays.

c) **MouseManager class added.**
Shop was not implemented in the first iteration, so in the first iteration the controls were done by the keyboard, but as we promised we implemented the mouse too. This class implements MouseListener and MouseMotionListener and it tells the Control subsystem when the mouse is clicked and the position. We use the mouse in the shop menu and in the other menus too. Without the mouse the user cannot access the Play button so it is a main component of the game.

d) **ShopManager class added.**
ShopManager class is added to provide an upgrade system for skills and allies. The game gets most of the attributes about the GameObjects from this class, like the maximum health, damage outputs, mana and food requirements.

4.2 Changes in GameModel subsystem.

a) **HailStrike class added.**
As the name implies the class function is to provide the animation for the strike based on the CollisionManager class that returns the objects “fights” based on the coordinates.

b) **HealSkill class added.**
Some objects in the game have the chance to heal the nearby allies, and in order to do that this classes was needed. Here we take care of the coordinates and check for the nearby allies. Only the ones at a certain range will get healed. The ones that are far away from the center of healing will not benefit anything.

c) **MonkeyAttack class added.**
Our monkey has a funny way of attacking and this class provides the animations and a pure projectile logic for it. Since, the monkey checks collision with a certain range, it could not have simply decrease the health of the targeted enemy. So, what we did was to implement this class which will work as an any other ally object does, but is removed from the game once the projectile hits the targeted enemy

d) **RavenStrike class added.**
Similar to the above class this one takes care of the RavenStrike projectile attack.

e) **SpeedBuffSkill class added.**
As the name implies this class simply increases the movement speed of the ally objects temporally.

4.3 Changes in UIManagement subsystem.

a) **ClickAction interface added.**
This İnterface was created for the onclick method that we will need in the control subsystem of the application.

b) **UIComponent abstract class added.**
This class has an instance object of GameManager class and it is used for writing or painting different components on the screen such as buttons or the background.

c) **UIButton class added.**
This class extends UIComponent class and it generates the image for the button in the main screen of the program.

d) **UIImage class added.**
This class extends UIComponent class too and it simply calls the parent methods to draw the image in the screen.

e) **UIBackground class added.**
This class extends UIImage class and it generates the background of the game by calling the super constructor of its parent.

f) **UIManager class added.**
UIManager class is the main class for the above explained classes if we can call it like that. It keeps an ArrayList of UIComponents and renders their graphics onto the main frame of the application.

5. What is left?

Nothing left from what we promised.

6. Conclusion

The purpose of the final report is to document the implementation process, deviation from design report and the incomplete parts. A user guide was also provided. In implementation part the Model, View and Controller are divided in subsystems. The Model subsystem is based on the GameObject class. The View will be based on the GameEngine class and controller class which is GameManager. In the deviation from design report the changes what we have done is explained.

We chose this project in order to learn more about 2D programming in Java and have a closer look at complex systems like the one we are making. By the time we finish our game, we will have learnt a lot of things about Object Oriented Software engineering. The idea was also to understand how to cooperate in a team-work, share ideas and choose together what is the best solution for a given problem that can occur. The implementation with the MVC is showing us its pros and its cons with the other types of implementations that exist nowadays, and in overall the project was the best chance we had to improve our OOP skills. We have all benefited from this project.

7. References

Sprites are taken from (1) and (2), Background image is taken from (3) and the background music is taken from (4)

[1] “Heroes of Might and Magic 2.” The 3DO Company, 1996.

[2] “2D GAME MONKEY CHARACTER SPRITE” Craftpix, https://craftpix.net/product/2d-game-monkey-character-premium-sprite/

[3] “Druid Wallpapers.” WallpaperCave, wallpapercave.com/druid-wallpaper.
[4] “Metin2.” Gameforge,“ https://tr.metin2.gameforge.com/