Bilkent University

Department of Computer Engineering

Object Oriented Software Engineering

CS 319 Project: The Hungry Beast

Final Report

Eren Erol
Hande Özyazgan
İrem Ergün
Yalın Eren Deliorman

Course Instructor: Özgür Tan

7.12.2015

Table of Contents

1. Introduction

The Hungry Beast is an arcade-style score based game. There is only one player who directs the Hungry Beast and tries not to stay hungry by shooting the foods and eat them. Foods will be disintegrated into pieces and the beast tries to catch them by moving horizontally so as to eat them. If the player cannot eat for a limited time, beast's health will decrease over time and the game will be over. During game, player will gain score according to the time spent and food eaten. The more the player spends his/her time while playing, the more he/she gains score accordingly. Since the game is score based, the goal in the game is catching as far as possibly pieces. In order to get a high score, beast should not be hungry for a long time. When the game is over, high scores will be saved to the high score table and player could check whether or not his/her score is in it.

There are 2 different game modes. One of them is called vs Creatures mode. In this mode, after 30 seconds after playing the game, in addition to normal pieces that feed the beast, there will be also creatures which is fallen from above and decreases beast's health according to creatures' level. Based on creatures' level, beast's health can be decreased less or more. Other game mode is vs Time that there will not be such creatures; player just tries to eat object drops from the above.

Our aim while creating this game is doing our version of arcade shooting game. By adding Bilkent University's instructors as creatures, it will be more exciting for a Bilkent student. Our difference between the other arcades shooting games is that the excitement for seeing our instructor's in a game.

Throughout the report, a detailed analysis of the application we are planning to develop is given. After reading this report, one can become quite informed about the application we are going to implement and the requirements we need to address to fulfil the expectations. Firstly, we will talk about the functional and nonfunctional requirements and constraints to clarify the expectations. Then, we will introduce scenarios so that the functional requirements are totally understandable. Also, the user interface of the game will be explained with visuals so that it is visible and more understandable. After that, the report will become more complex and it will

appeal to the developers since we will explain the requirements in detail by getting help from UML modeling.

2. Requirement Analysis

This part includes overview, functional requirements, non-functional requirements, constraints, scenarios, use case models, user interface of the Hungry Beast. This part addresses to both developers and the clients. Hence, it is like a bridge between the developers and the users of the application.

2.1 Overview

Beast: Beast is the user controlled object which has the ability to shoot the objects above it using the weapon it carries. After shooting to the objects that are above itself, it tries to catch the food that comes down from those objects. Beast has a decreasing health property which is refilled after it catches and eats the food.

Weapon: Weapon is the tool that the beast uses the shoot the objects. There are three types of weapons in the game. These types are the following:

1. Wood throwing gun: It is the most primitive weapon type and the user has only this type of weapon initially. It gives 5 damage to the shot object. Its cooldown time is 0.5 second.
2. Bullet throwing gun: It is the second weapon that the user unlocks after passing some score threshold. It gives 20 damage to the shot object. Its cooldown time is 1 second.
3. Laser gun: It is the most advanced weapon type in the game. The user has the chance to unlock it after he/she has unlocked the bullet throwing gun. It gives 30 damage to the shot object. Its cooldown time is 1 second.

Food: These will increase the health of the beast after being caught by the beast. Food needs to be split into pieces. There will be three types of food. They are given below:

1. Sugar: Increases health points by 5 points. The user needs to give 50 damage to sugar object in order to split it into pieces. After being split, sugar will be shredded into five pieces and each piece will give 1 health points to the beast. (Total of 5 points)
2. Vegetable: Increases health points by 10 points. The user needs to give 140 damage to vegetable object in order to split it into pieces. After being split,

vegetable will be shredded into five pieces and each piece will give 2 health points to the beast. (Total of 10 points)

3. Meat: Increases health points by 30 points. The user needs to give 300 damage to meat object in order to split it into pieces. After being split, meat will be shredded into five pieces and each piece will increase the health points of the beast by 6. (Total of 30 points)

Creatures: These will decrease the health of the beast if they touch the beast. User needs to control the beast by making it avoid all of the creatures. There will be three types of creatures. Creatures are invulnerable. They are the following:

1. DavidDavenport: Decreases health point by 25 points.
2. WilliamSawyer: Decreases health point by 50 points.
3. OkanTekman: Decreases health point by 75 points.

How to Play the Game

The game screen is divided into three lanes. The user starts to play in the middle lane and uses arrow keys to move to the right or left lanes. When the user presses space, the beast shoots.

The beast initially has 100 health points. Health points of the beast start to decrease. The more the user proceeds in the game, the faster the health points of the beast will decrease. For example, in the beginning, the beast loses 1 health point for every 3 seconds during which it does not eat any food. The beast gains health according to the level of the food it has eaten.

Score

There is a timer in the game which keeps track of the time. The value in the timer will determine the score. Top 3 scores will be kept in the high score table.

Game Modes

There are two different game modes:

vs Time:

In this game mode, the user will only hit the objects above and try to catch the food that are coming from them. In this mode, the aim of the user is to survive as long as possible. The longer it survives, the higher score he/she gets.

vs Creatures:

After timer exceeds 30 seconds, creatures that have the ability to decrease the beast's health points will come to the lanes. If those creatures fall onto the beast, beast's health points will decrease according to the level of the creature. (25 points for DavidDavenport, 50 points for WilliamSawyer, 75 points for OkanTekman)

2.2 Functional Requirements

The Hungry Beast has game modes, pause/resume game, tutorial, high score, credits and change settings as functional requirements.

Game Modes

The Hungry Beast has two game modes. In the first mode user plays the game against the clock and tries to get maximum score. In the second mode, user tries to keep monster alive, so after 30 seconds, creatures falls down and when they touch to the monster, the life time of monster decreases.

Pause/Resume Game

Player may need to pause the game from any reason and for not losing all scores which he/she has, The Hungry Beast game offers pause game functionality. Without exiting from game, player can pause the game and then resume on.

Tutorial

For the first time of the user playing the game, The Hungry Beast offers a tutorial to teach the player dynamics and rules of the game. In tutorial, players learns the general information about the game, purpose of the game, difficulty levels, control keys and how to command the creature and game rules.

High Score

Main menu provides an option for showing the high score table. The table will show the high scores of the game which will be determined by the time elapsed and food eaten. Player can enter his/her name in the table.

Credits

Credits screen shows brief information and contact information of creators of the game and user can access credit screen through main menu.

Change Settings

The Hungry Beast manually uses left and right arrow keys for moving horizontally and space key for shooting. User can change the arrow keys to WASD control set for moving, where “a” moving to left and “d” moving to right. Also, the user can change the color of the user controlled object, Beast, from the ‘Change Settings’ screen.

2.3. Non-Functional Requirements

The Hungry Beast has performance, usability, maintainability as non-functional requirements.

Performance

The game is played with only one user. There will be minimum latency as much as possible. The monster’s size are much bigger than the foods to make the game a little bit difficult. It does not use a huge amount of memory. The response time of the application will be kept at minimum in order to improve the user experience quality of the application.

Usability

Our game favors simplicity. Hence, understanding the game dynamics and how to play is easy for every user. Controls of the game are not hard because the user only uses three buttons on the keyboard to play the game. User also can change the controls in the settings if he/she prefers to use WASD control set instead of arrow keys. If the user wants to understand the game before even starting to play, then he/she can access the ‘Tutorial’ screen, which is one click away from the main menu. In ‘Tutorial’ screen, a detailed explanation of the dynamics and objects of the game is provided.

Extensibility

Users get bored after playing a game several time. Therefore, extensibility is very crucial in game applications. So, we will design our system in a way that it is easy to extend. We will have separate views. Hence, it will be possible to extend it easily by modifying the views and creating new associations.

Portability

For our game to be portable, we will implement it in Java. So, regardless of the operating system of the computer the game is installed in, the application will run.

2.4 Constraints

- The program will be written in Java. So, as to run the program, JDK should be installed in the computer.
- This game is a desktop application. It does not need to be implemented in mobile environments.
- High score data should be stored in a .txt file for convenience.

2.5 Scenarios

How to play Versus Time Mode

The user is shown GamePanel screen, which is the main menu of the application. After user selects 'Play Game' option in the main menu, he/she is shown the viewMode screen of the game and user sees the modeList of the program. There are two different game modes in the list, which are Versus Time and Versus Creatures. The user then selects the 'Versus Time' mode and pushes the 'Play' button. After user takes these actions, game initializes 'versus time' mode and user starts to play. Initially, a beast and a weapon object will be created for user to interact with the application. Throughout the game, several food objects will be created and destroyed by the user. When a food object is destroyed, it will produce several shredded food objects. Whether there is a collision or not between the beast and the shredded food objects ischecked during the game. If there is a collision, health of the beast increases according to the level of the food that was shredded. The health of the beast the user controls decreases within a specific time interval and health points of the beast is checked throughout the game. If the health points of the beast becomes zero, the game will be terminated. The score is given according to the time that the beast object survives during the game. The longer the beast survives, the higher the score of the player is. After the game is terminated, the score of the user is calculated according to the survival time. If he/she gets an high score, the application asks the name of the user and gets it from the user. Then, that score is shown in the high score table.

How to play Versus Creatures Mode

The user is shown GamePanel screen, which is the main menu of the application. After user selects 'Play Game' option in the main menu, he/she is shown the viewMode screen

of the game and user sees the modeList of the program. There are two different game modes in the list, which are Versus Time and Versus Creatures. The user then selects the ‘Versus Creatures mode and pushes the ‘Play’ button. After user takes these actions, the program initializes ‘versus creatures’ mode and user starts to play. Initially, a beast and a weapon object will be created for user to interact with the application. Throughout the game, several food objects will be created and destroyed by the user. When a food object is destroyed, it will produce several shredded food objects. Whether there is a collision or not between the beast and the shredded food objects ischecked during the game. If there is a collision, health of the beast increases according to the level of the food that was shredded. Also, there are creatures in the game which will reduce the health points of the beast in case of collision. The health of the beast the user controls decreases within a specific time interval even though the beast does not collide with a creature object and health points of the beast is checked throughout the game. If the health points of the beast becomes zero, the game will be terminated. The score is given according to the time that the beast object survives during the game. The longer the beast survives, the higher the score of the player is. After the game is terminated, the score of the user is calculated according to the survival time. If he/she gets an high score, the application asks the name of the user and gets it from the user. Then, that score is shown in the high score table.

Firing the weapon

The user is shown GamePanel screen, which is the main menu of the application. After user selects ‘Play Game’ option in the main menu, he/she is shown the viewMode screen of the game and user sees the modeList of the program. There are two different game modes in the list, which are Versus Time and Versus Creatures. The user then selects the desired game mode and pushes the ‘Play’ button. After user takes these actions, game initializes itself according to the chosen mode and user starts to play. When the user presses space button (not during the cooldown period of any weapon), the weapon of the beast will fire missiles and decrease the health points of the food objects in case of collision with any missile. The health points of the food object will be checked and if it becomes zero, that food object will be terminated. The application keeps checking the health points of the beast until it becomes zero and the game gets terminated. Before termination, the user has the ability to make the beast fire missiles as many times as he/she wants. After termination, the game checks whether the score of the user is a high

score. If it is, the game asks the name of the user and gets it. Then, the game displays it in the high score table.

2.6 Use Case Models

Play game, pause game, change settings, view tutorial, view high score and view credits use cases are the use cases.

Play Game

Use Case Name: Play Game

Primary Actor: Player

Goals and Interests:

-Player wants to play the game.

-Player aims to survive as long as possible and get the highest score.

Entry Condition: Player chooses to play one of the game modes.

Exit Condition: The game is over which means the health of player is 0.

Success Scenario Event Flow:

1. Game provides two game modes to the player.
2. Player selects a game mode among two modes.
3. System provides the main screen for playing the game.
4. Player tries to stay alive as long as possible.
5. System shows score of the player.
6. If the player's score is in the top 3 scores for that game mode, system will ask for a username for high score table.
7. System goes to step 1.

Alternative Flows:

4.1.1 Player presses to the pause button.

4.1.2 System switches to "Pause Game (4.1.2)" case.

Pause Game

Use Case Name: Pause Game

Primary Actor: Player

Goals and Interests:

- Player wants to pause the game.
- Player wants to restart the level.

- Player wants to exit the game.

Entry Condition: Player must press Pause Game in the game.

Exit Condition: Player chooses to resume game, restart game or exit to main menu.

Success Scenario Event Flow:

1. System presents resume game, restart game or exit to main menu options to player.
2. Player presses resume button.
3. System makes user back to the game.

Alternative Flows:

1.1.1 Player chooses to restart.

1.1.2System restarts the game.

1.2.1 Player choses to exit to main menu.

1.2.2 System goes to main menu.

Change Settings

Use Case Name: Change Settings

Primary Actor: Player

Goals and Interests:

-Player wants to change gameplay settings.

Entry Condition: Player presses settings button.

Exit Condition: Player finishes changing settings.

Success Scenario Event Flow:

1. System presents keyboard settings.
2. Player change keyboard settings.
3. Player is done with changing settings.
4. System saves state of the settings.

Alternative Flows:

There is no alternative flow.

View Tutorial

Use Case Name: View Tutorial

Primary Actor: Player

Goals and Interests:

-Player wants to learn how to play the Hungry Beast.

Entry Condition: Player chooses to see tutorial.

Exit Condition: Player finishes reading tutorial.

Success Scenario Event Flow:

1. System shows the tutorial of the game to player.
2. Player indicates that he is finished.

Alternative Flows:

There is no alternative flow.

View High Scores

Use Case Name: View High Scores

Primary Actor: Player

Goals and Interests:

-Player wants to see the high score table.

Entry Condition: Player chooses to see high score table.

Exit Condition: Player finishes checking high score table.

Success Scenario Event Flow:

1. System presents player high score table.
2. Player exits the high score table.

Alternative Flows:

There is no alternative flow.

View Credits

Use Case Name: View Credits

Primary Actor: Player

Goals and Interests:

-Player wants to learn about the developer team.

Entry Condition: Player chooses to see credits.

Exit Condition: Player finishes seeing credits.

Success Scenario Event Flow:

1. System presents information about developers.
2. Player indicates that he is finished.

Play vs Time Mode

Use Case Name: Play vs Time Mode

Primary Actor: Player

Goals and Interests:

-Player wants to play the vs time mode of the game.

Entry Condition: Player presses ‘Play Game’ button on the main menu.

Exit Condition: Player terminates the game after pausing or player loses the game (in other words, health points of the Beast object in the game becomes zero).

Success Scenario Event Flow:

1. Player presses ‘Play Game’ button on the main menu.
2. Player selects the game mode by pressing ‘vs Time’ mode and pushes the ‘Play’

3. Beast object is created and user controls it.
4. Health points of the beast object decreases every passing second.
5. User moves the Beast object right and left.
6. User shoots the food objects above and tries to smash them.
7. User shreds the food object and collects the shredded foods to gain health.

Play vs Creatures Mode

Use Case Name: Play vs Creatures Mode

Primary Actor: Player

Goals and Interests:

-Player wants to play the vs creatures mode of the game.

Entry Condition: Player presses ‘Play Game’ button on the main menu.

Exit Condition: Player terminates the game after pausing or player loses the game (in other words, health points of the Beast object in the game becomes zero).

Success Scenario Event Flow:

1. Player presses ‘Play Game’ button on the main menu.
2. Player selects the game mode by pressing ‘vs Creatures’ mode and pushes the ‘Play’

3. Beast object is created and user controls it.
4. Health points of the beast object decreases every passing second.
5. User moves the Beast object right and left.
6. User shoots the food objects above and tries to smash them.
7. User shreds the food object and collects the shredded foods to gain health.

8. After 30 seconds, creatures start to show up and the beast object tries to avoid them.

Alternative Flows:

Power Out

This is an exceptional use (not a success scenario) case happens only if the electricity of the computer that the user plays goes off. In that case, the game is not saved and the user needs to start all over. The game data is not saved in this situation.

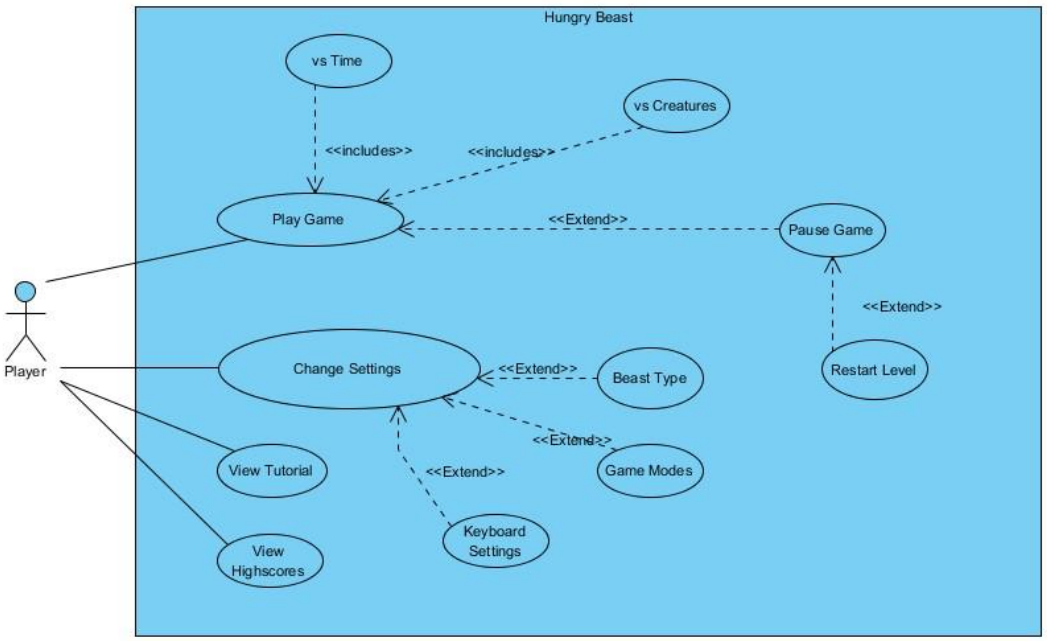

Figure 1: Shows the use-case diagram of the Hungry Beast.

2.7 User Interface

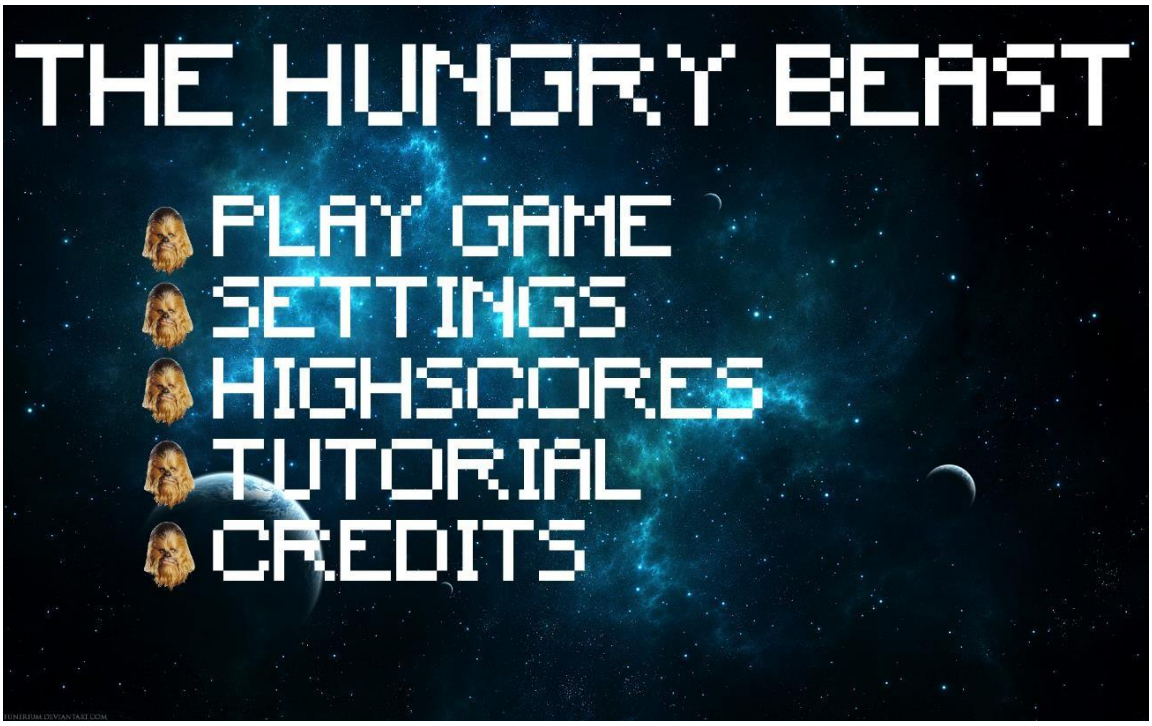

Figure 2: Shows the main menu of our game.

Main menu is the first screen that shows to the player. In main menu, player can start the game and choose the game mode or view high scores, Credits, Settings and Tutorial from here.

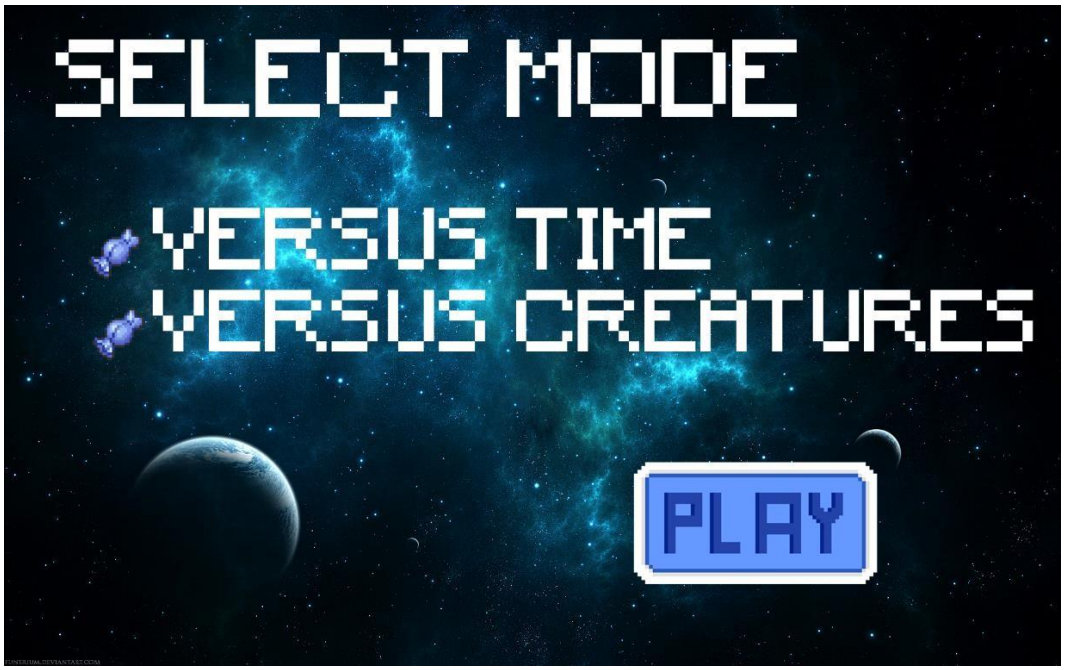

Figure 3: Shows the screen after choosing play game button.

Player can choose among from two modes. After clicking the mode that he/she wishes to play, he/she must click Play button for to start the game.

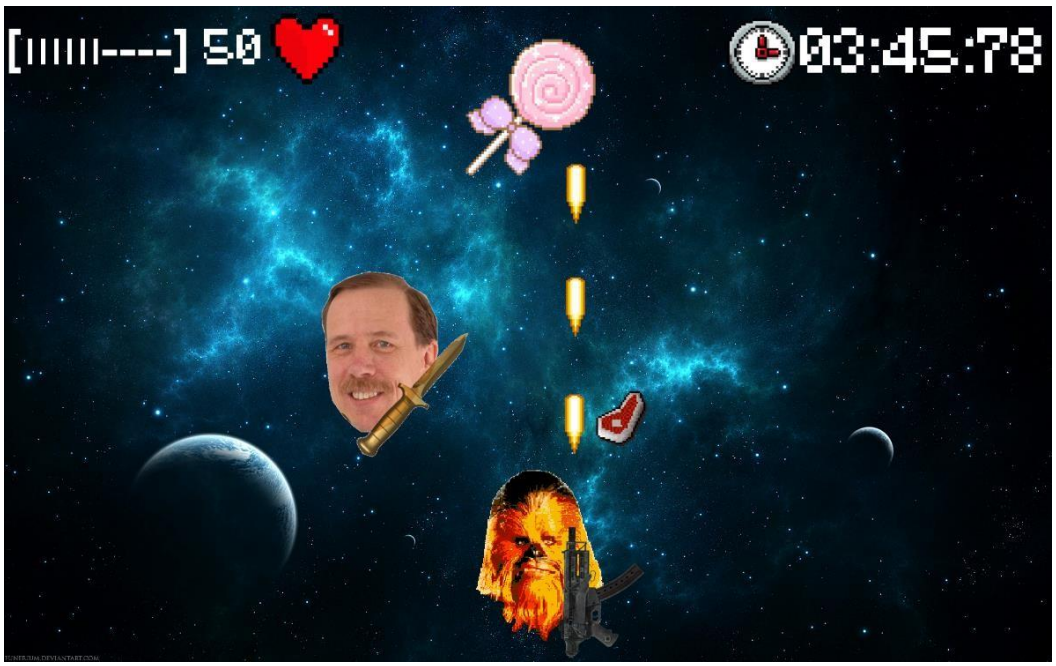

Figure 4: Shows the gameplay of the Hungry Beast.

This is a gameplay screen capture of vs Creatures mode. Hungry beast is shooting with its weapon. In the same time random food and creature object are dropping. Health and timer can be seen from the above.

Figure 5: Shows the change settings screen.

From the change settings screen that can be reached from both during gameplay and main menu, there are three selectable different beast types. Each one has a different weapon. Game mods and keyboard settings can be adjusted from here too.

Figure 6: Shows the high scores name entering screen.

After playing and finishing any game mod, if player's high score is high enough to enter the high score table, system will ask a name from the player. Player can enter his name from here. This page will be directed to high scores table.

Figure 7: Shows the high scores table with the corresponding game mode.

After entering the name, user will be directed to high scores table. Each game modes has its own high score part for keeping the track of records. Using can return to main menu by clicking the main menu button.

If the user wants to learn about the game before starting to play, he can click ‘Tutorial’ button from the main menu and be directed to Tutorial screen. In this screen, user can find some information about the general logic and the objects of the game.

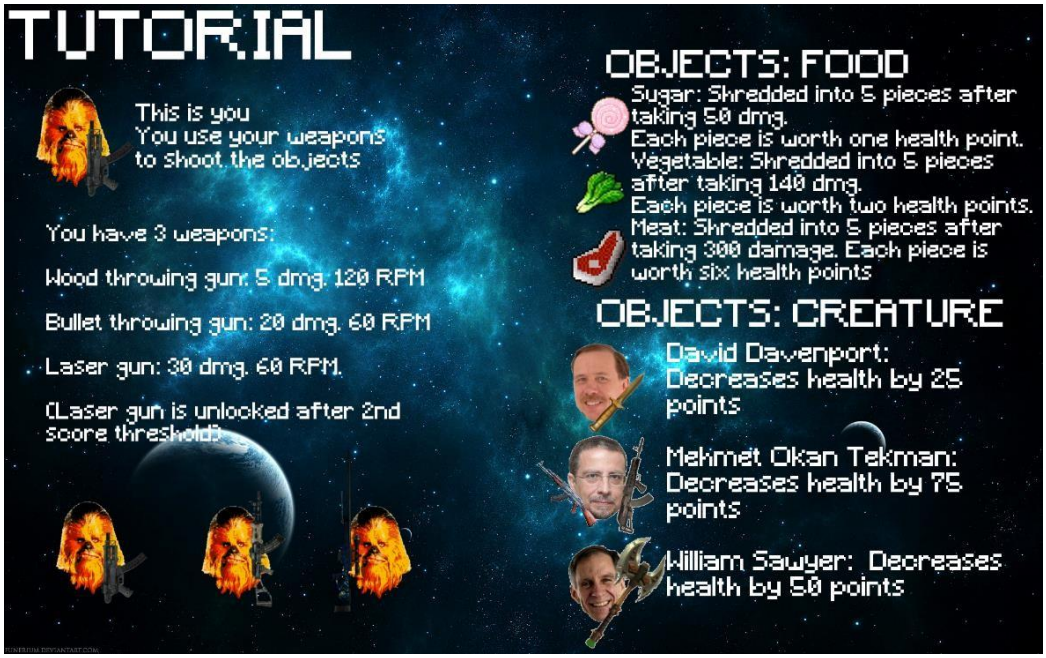

Figure 8: Shows the tutorial screen

Also, credits can be found in the main menu. User can press the ‘Credits’ button and be directed to Credits screen. In this screen, some information can be found about the people who contributed to the development of the application.

Figure 9: Shows the credits screen

Foods: These are the three different food types that will appear during gameplay. First one is sugar, second is meat and the third one is vegetable. All of them has different attributes as it is mentioned before.

Figure 10: Shows the food types in the game.

Creatures: These creatures will only appear in vs Creature mod. Each of them is reducing amount of health points from the beast. They are invulnerable.

Figure 11: Shows the creatures in the game.

3. Analysis

The domain lexicon, class diagram, state chart diagram and sequence diagrams are covered in this part.

3.1 Object Model

3.1.1 Domain Lexicon

Beast: This is an entity class for the user who plays the game. In the game, user directs the beast and tries to shoot foods above it. While the user is choosing the beast, he/she can change its color. Additionally, each beast has health attribute and either or not game is over will be decided based on its health.

Food: This class is an entity which reflects the real foods in the game. When the user manages to shred food, food will be split into number of pieces. Each food has health and when the food is split, each piece has also health according the

number of pieces. There will be some kind of foods, such as sugar, vegetable and meat. Therefore, this is an environment that consisted of other entities like Sugar, Vegetable and Meat.

Creature: This class is an entity which affects the beast's health negatively. In order to make the game a little bit difficult, there should be some creatures, some of which are DavidDavenport, WilliamSawyer and OkanTekman. Each creature negatively affects the beast's health differently.

Weapon: This class is an entity for real weapons. In order to decrease foods' health and split those into pieces user should use weapons. There will be cooldown so that user cannot shoot continually. There will be three different weapons some of which are Wood, Bullet and Laser. In terms of damage given by weapons, they are grouped like that. User plays the game with wood firstly, then, when user gets ahead in the game, he/she will be able to use bullet and laser.

3.1.2 Class Diagram

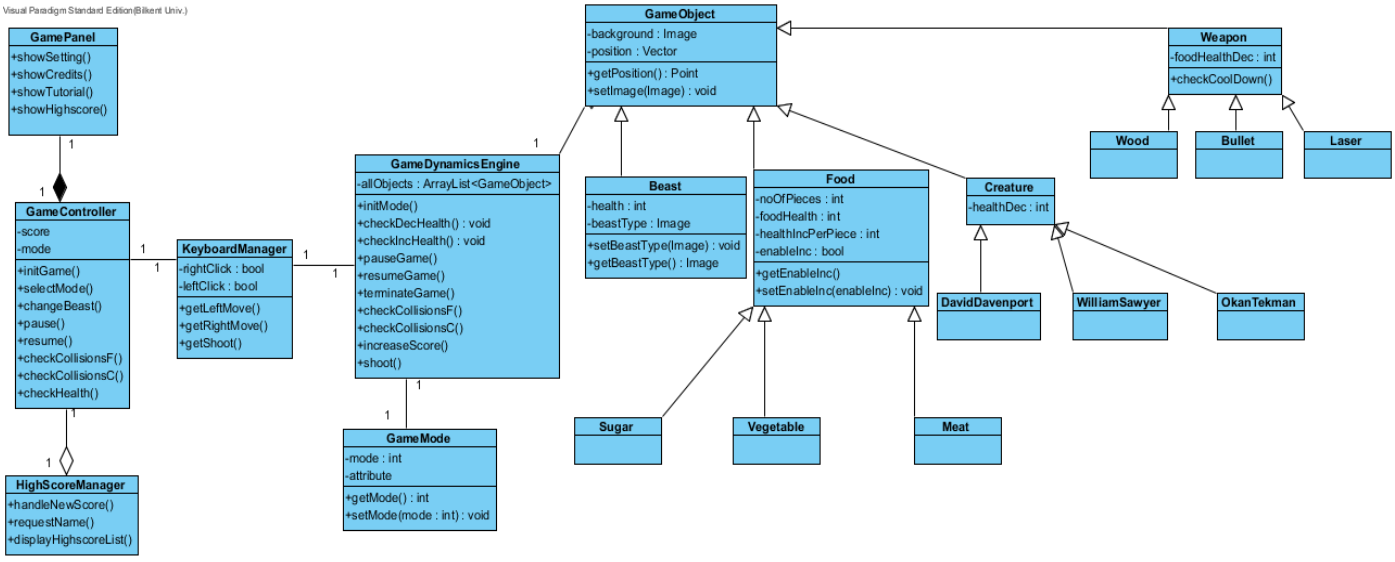

Figure 12: Shows the class diagram of the Hungry Beast.

In our design we have 20 classes. GamePanel class handles the GUI part of the program. GamePanel is interacting with the GameController class which is responsible initializing the game, selecting mode, changing the beast and resume/pause the game. It also have score and mode as properties to keep track of them. GameController class is related to HighScoreManager and KeyboardManager. HighScoreManager keep the track of high scores and the owner of these high scores. KeyboardManager is interacting with the GameDynamicsEngine class which is responsible for the fundamentals of the game. These are checking the collision between food and beast (checkCollisionsF()) also between beast and creature (checkCollisionsC()). There are methods also for controlling checking the changes in beast's health, which are checkDecHealth() or checkIncHealth(). GameDynamicsEngine is related with 2 GameMode classes, vs Time and vs Creatures modes.

The remaining classes are all related with the GameObject class. Beast, Food, Creature and Weapon classes are inheritably combined with the GameObject class as children. Beast class has health and color as its properties. Food Class has three children which are Sugar, Vegetable and Meat classes. After foodHealth is 0, setEnableInc(enableInc) method will be true. This allows food to shred into noOfPieces. Each different type of food will give different healthIncPerPiece. Creature class has DavidDavenport, WilliamSawyer and OkanTekman classes as children. These creatures only have healtDec property which will differ by class. Weapon class has foodHealthDec property and checkCoolDown() method. Wood, Bullet and Laser which are 3 children of Weapon class will have different cooldown times and different food health decreasing amounts.

3.2 Dynamic Models

Dynamic models part have state chart diagram of the Hungry Beast and three different sequence diagrams which are responsible for showing two different game modes and firing a weapon.

3.2.1 State Chart

State chart diagrams are useful to depict the dynamic behavior of a single object. Below, the behavior of two fundamental objects of the game, the user controlled object, Beast, and the food object are shown using state chart diagrams.

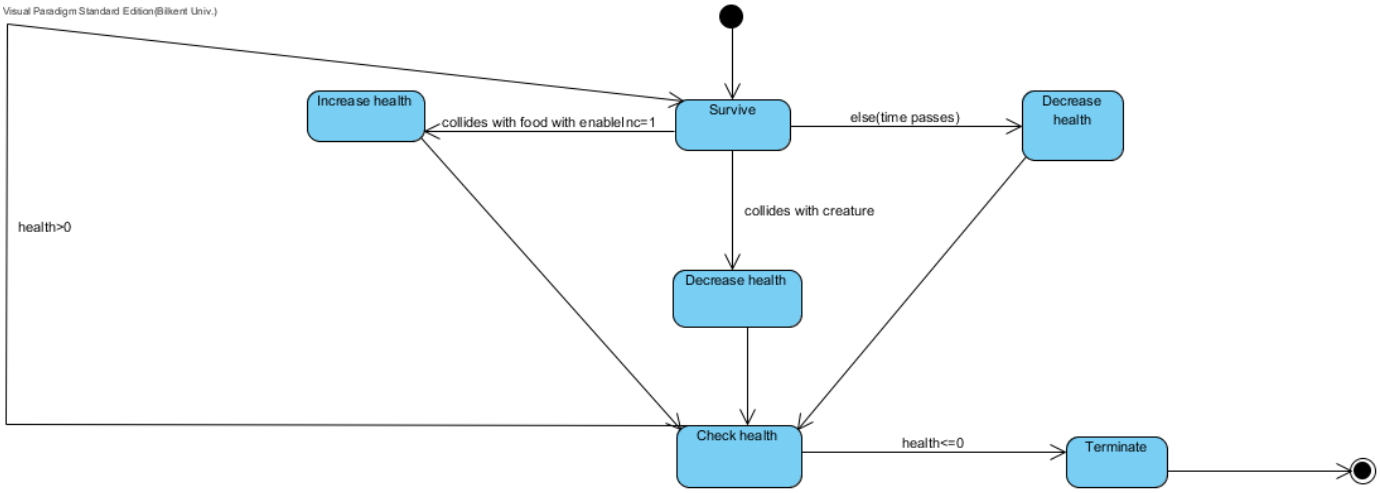

Figure 13: Shows the state chart diagram of the Beast object.

After the creation of the beast, its health points decrease as the time passes. Also, beast loses health points if it collides with a creature and beast gains health points if it collides with a food object with enableInc=1 (shredded food object). The diagram below depicts response of the beast object to these events. When there is a change in the health points of the beast object, the health points of the beast is checked without a transition. If it is greater than zero, the beast goes back to the 'Survive' state and its health points is checked again when a change occurs.If the health points of the beast becomes less than or equal to zero, the game terminates.

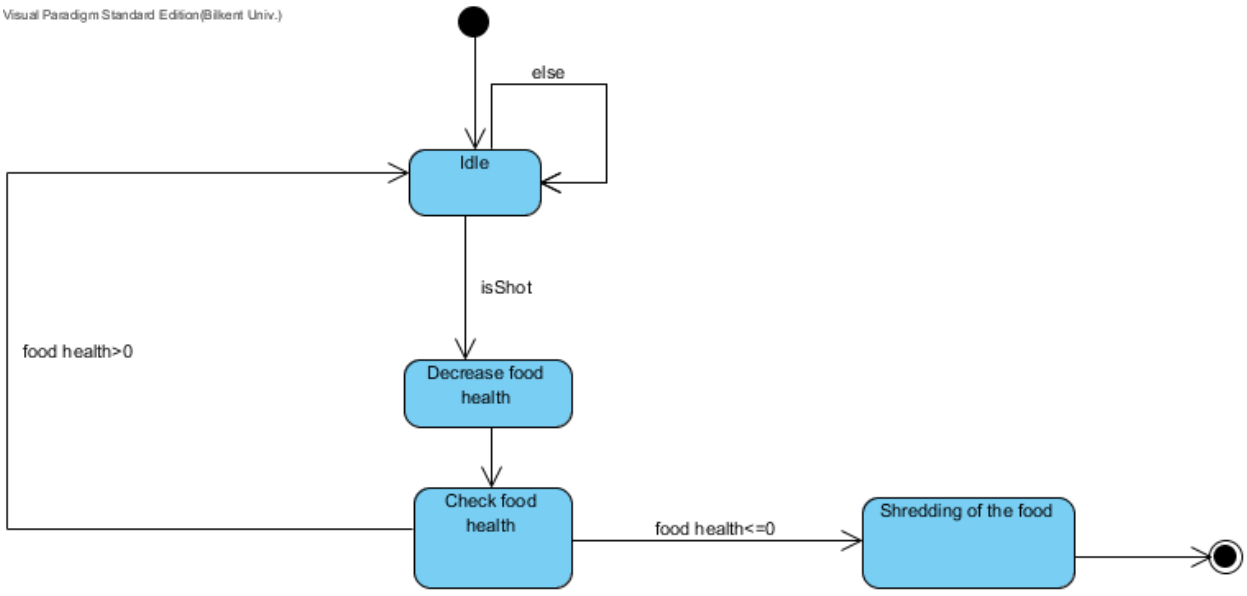

Figure 14: Shows the state chart diagram of the Food object.

If the food object is not shot, there is not a change in its state. If it is shot, then its health points decrease accordingly and then its health points are checked. If health points of the food is greater than zero, then the food goes back to the 'Idle' state and its health points is being checked again. If it is less than or equal to zero, food is shredded according to its level. Then, the food object becomes shredded food object (enableInc becomes 1) and it does not go back to any other state. Then, the diagram shows the termination of the food object after it is being shred.

3.2.2 Sequence Diagram

Playing in vs Creatures mode: Player wants to play vs Creature mode in Hungry Beast. Firstly, he/she views available modes in the game. He/she chooses vs Creatures mode. System selects the mode and initialize mode and beast. The game mode will appear in the screen after playing the play button. Beast's weapon is initialized. After the game starts food will randomly drop. For every destroyed Food object beast will gain health. After 30 seconds in the game, Creature objects starts to fall from above in random time intervals. After the game is finished, player's high score is checked whether it is qualified enough to enter the high score table. In this scenario it is enough. System asks a name from the user. After entering the name, system shows high score table to player.

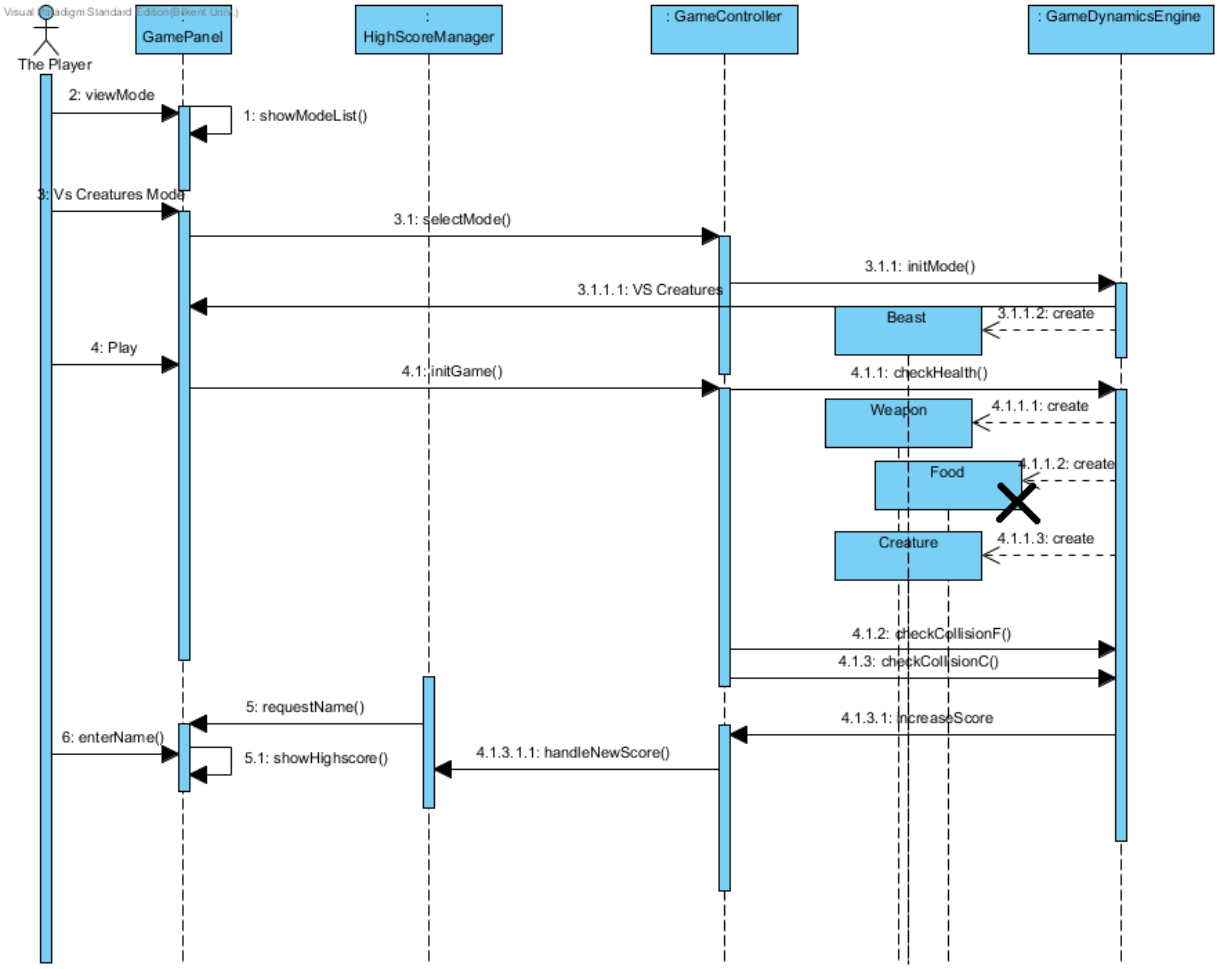

Figure 15: Shows a scenario for game mode vs Creatures.

Playing in vs Time Mode: Player wants to play vs Time modes in Hungry Beast. Firstly, he/she views available modes in the game. He/she chooses vs Time mode. System selects the mode and initialize mode and beast. The game mode will appear in the screen after playing the play button. Beast's weapon is initialized. After the game starts food will randomly drop. For every destroyed Food object beast will gain health. The only difference in this game mode from vs Creatures is there are no creatures that decreases life. Therefore, the game can only end, if player cannot eat enough food before its health drop to 0. In this scenario, player is qualified to enter the high score table too. System will ask for his/her name and will display on the high score table.

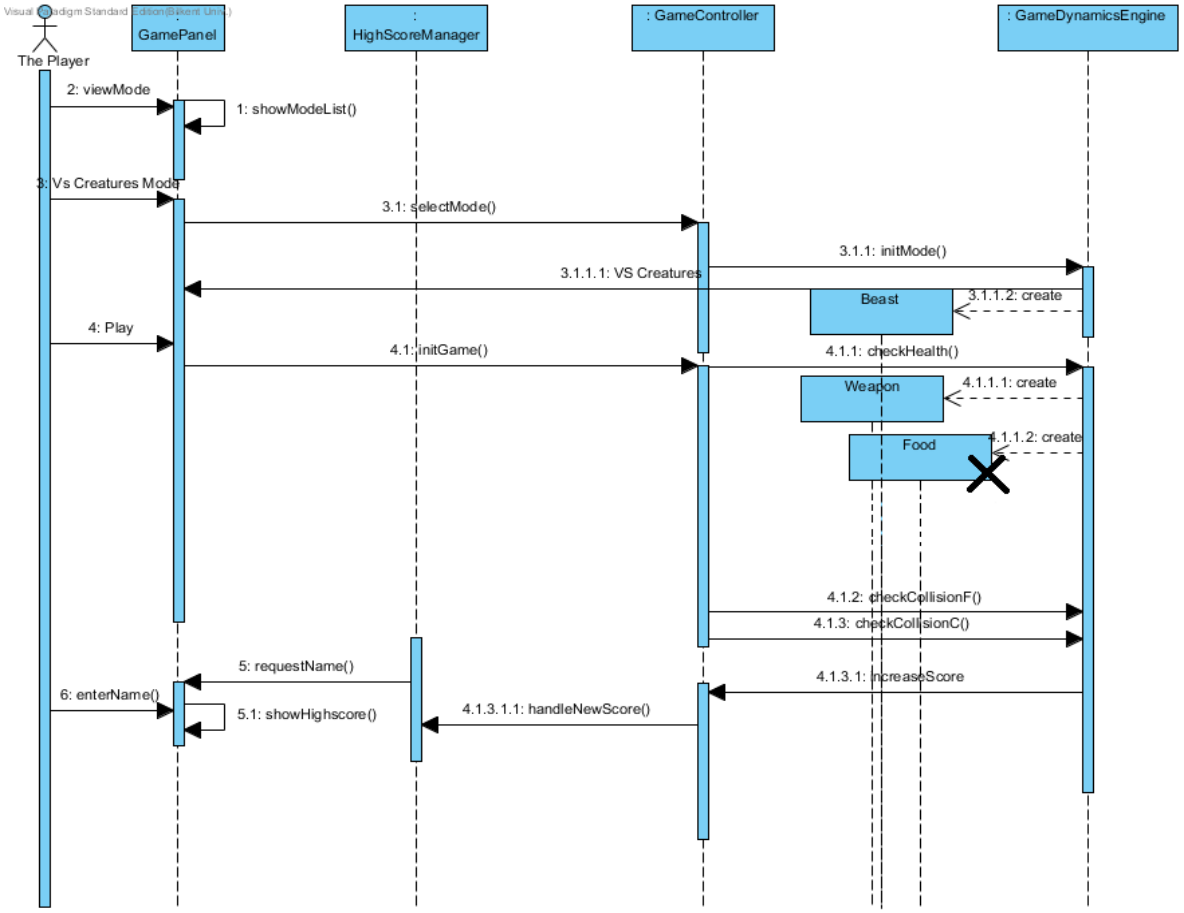

Figure 16: Shows a scenario for game mode vs Time.

Firing the Weapon: In this scenario, player wants to fire beast's weapon. Firstly, he/she views available modes in the game. He/she chooses vs Creatures mode. System selects the mode and initialize mode and beast. The game mode will appear in the screen after playing the play button. Beast's weapon is initialized. Until this part, it is same as the vs Creature mode. Player presses space for firing the weapon. It checks weapon cool down time. If it is available, weapon shoots again. If food's health is less than or equal to 0, it will split in to pieces. When player cannot gather enough food, beast's health will fall below 0 and game will finish. He/she enters his/her name for high score table.

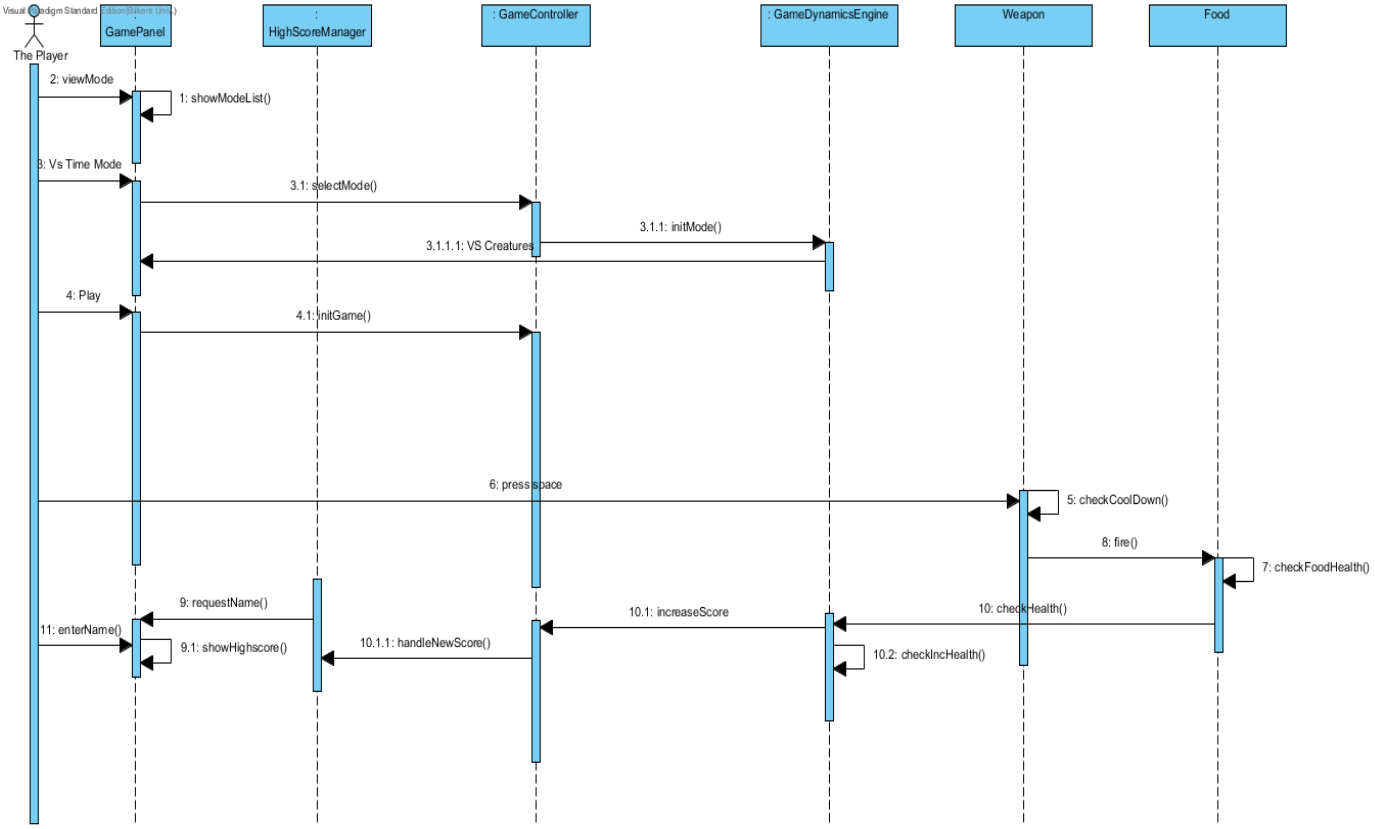

Figure 17: Shows a scenario for firing the weapon.

4. Design

The Hungry Beast is a “shoot ‘em up” game where the aim of the user is to shoot objects in the game and survive. However, it is not as easy as it sounds. Player is losing health points in every passing second. As the game proceeds, the health points decrease more rapidly and the user needs to gain more health from the shot objects to survive. But also, as the game proceeds, it is harder to shred the food objects and gain health accordingly. So, the player needs to be faster and more careful if he/she wants to survive. The game aims to improve the hand-eye coordination of the player and it is intended that The Hungry Beast also helps to improve the ability to focus on a specific task of its players.

4.1 Design Goals

End User Criteria:

Usability: We want the players of the game to learn how to play the game fast. Faster the user learns the game faster he/she will start enjoying the game. So, we are strong-minded about putting a well-documented tutorial that can be reached easily from the main menu so that the user will have a detailed knowledge about the game even before starting to play. We will also develop the game in a way so that a user who starts to play before reading the tutorial can understand and enjoy the game. Hence, we will keep the gameplay simple.

Utility: We strongly believe that this game will have a positive effect on improving the skills of many people. They will have the opportunity to improve their focus and hand-eye coordination.

Quality Graphics: We want our users to enjoy the game as much as possible. The better the graphics of the game are, the more the user wants enjoys the game. Even though we are using funny (but simple) icons to entertain the users, we will also pay attention to make the graphics as qualified as possible.

Dependability Criteria:

Robustness: Since we are developing a game, players will most likely give wrong commands in their learning phase. To avoid consequences of unwanted errors, we will give some importance to the robustness of the game.

Maintenance Criteria:

Extensibility: Users get bored after playing a game several time. Therefore, extensibility is very crucial in game applications. So, we will design our system in a way that it is easy to extend. We will have seperate controllers for each feature of the game. Hence, it will be possible to extend it easily by modifying the controllers and creating new associations.

Testability: Since we want to modify the game to extend its functionality, we need the system to be testable to detect the errors and improve them. For the application to be testable, it needs to be simple and available.

Portability: We want our game to be running on every platform regardless of the operation system or its version. Therefore, we will implement our game in Java. Every computer which has a JVM on it (in other words, whose user have installed Java), will run our game.

Performance Criteria:

Response Time: For the satisfaction of the users, a smooth gameplay is important. Users will get bored if the game does not respond to them fast. So, we will keep our response time as small as possible. We will try to create a system that reacts to user interactions immediately. 0.1 second is about the limit for having the user feel that the system is reacting instantaneously. So, we will try to keep the response time close to 0.1 second.

4.2 Subsystem Decomposition

The software is composed of three subsystems. These are User Interface Subsystem, Data Management Subsystem and Gameplay Subsystem. The criteria between these subsystems are maximum high coherence and low coupling.

User Interface Subsystem controls the user interaction with system by consisting of menu bar and information screen so it provides navigational functionalities to the user. User Interface Subsystem takes user input and transmits to the Gameplay Subsystem and shows the output of Gameplay Subsystem to the user.

Data Management Subsystem handles storing and accessing game settings, game mode and high score so the interface provided by Data Management Subsystem handles storing settings and retrieves game mode and data of high score which are basic text files.

Gameplay Subsystem presents the most essential functionalities because it manages all game actions. So Gameplay Subsystem provides initialisation of stages, creation and destruction of monsters, creatures, food and weapon in a harmonious way with user input so rules of two different stages. Gameplay Subsystem interacts with User Interface Subsystem for getting user input and interacts with Data Management Subsystem for getting game mode and settings.

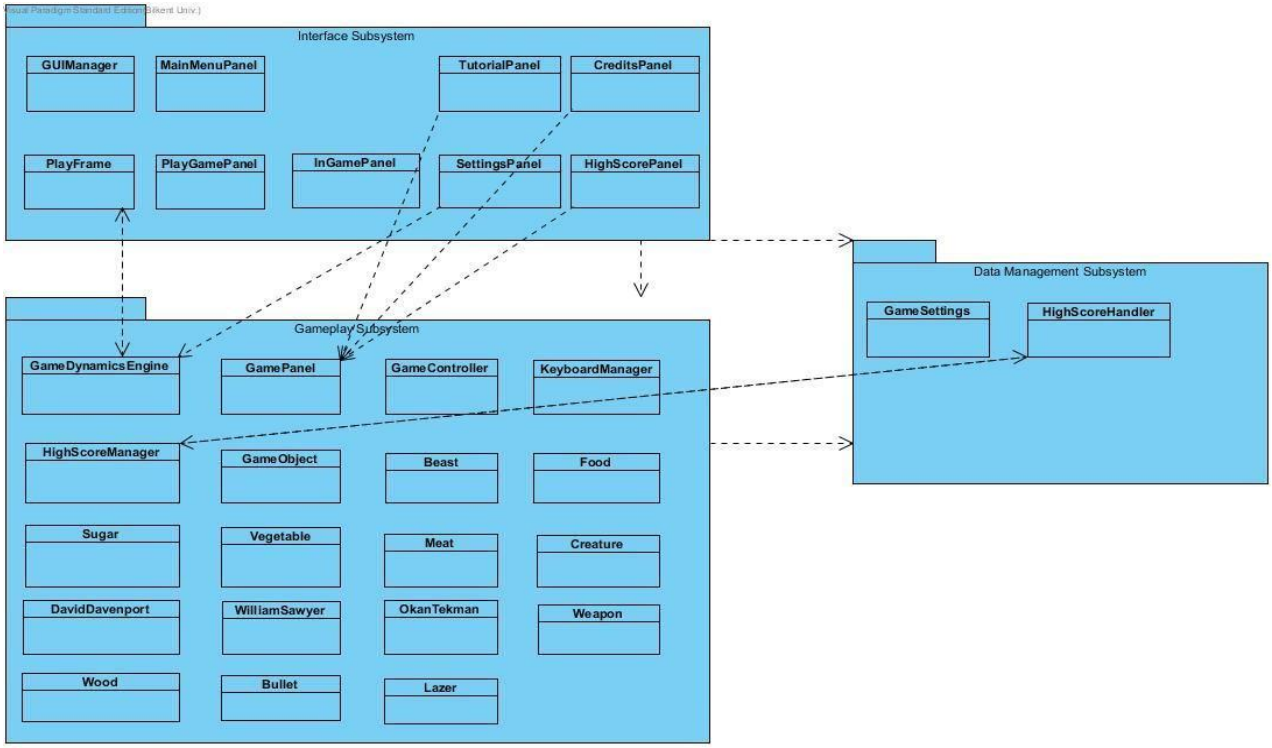

Figure 18: Shows the Subsystem relationships.

Below, whole GUI independant package of the Hungry Beast can be found. Associations and hiearchy can be seen from the below figure.

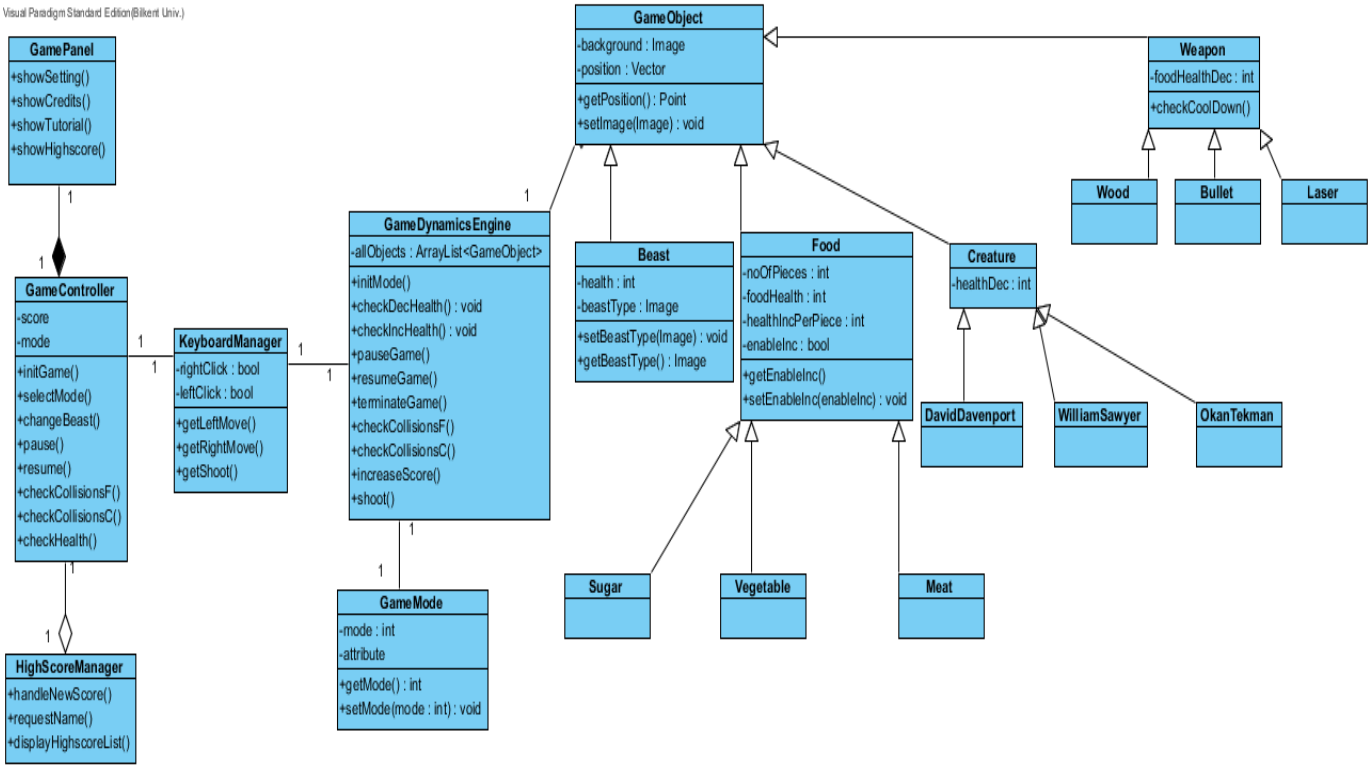

Figure 19: Shows the GUI independent package.

GamePanel has the ability to reach settings, credits, tutorial and high scores menu. GamePanel has a composition relationship with GameController. GameController keeps the score and game mode. It can also initialize the game, select the game mode, change beast, pause game, resume game, check collisions for food and creature and check health operations to keep the track of the game. HighScoreManager has an aggregation relationship with GameController. HighScoreManager handles the new high score, ask for a name and display the high score list to user. GameDynamicsEngine is related with KeyBoardManager and it does the all operations done by GameController. In addition to that, it keeps all game objects as an attribute and four extra operations which are shoot, increase score, check health increase, and check health decrease methods. GameDynamicsEngine is related with GameMode. It keeps the two game modes which are vs creatures and vs time. GameDynamicsEngine is related with GameObject. All game objects have an image and a position. GameObject has Beast, Food, Creature and Weapon classes which are explained before in the analysis report.

In below figure, attributes and operations of GUI package is represented.

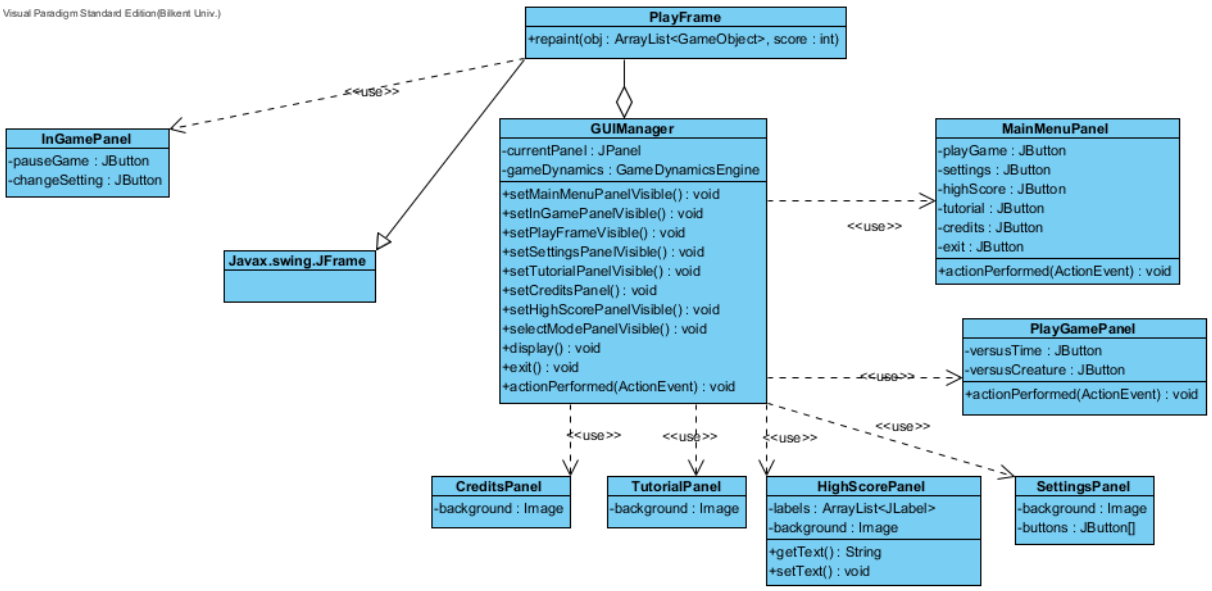

Figure 20: Shows the GUI package.

We have a PlayFrame class in order to keep the repaint method which does the GUI viewage part. PlayFrame uses InGamepanel which has pauseGame and changeSetting as Jbuttons. PlayFrame also has a Jframe class. PlayFrame has a aggregation relation with GUIManager. It has currentPanel and a gameDynamics engine instance as attributes. GUIManager makes panels

visible. GUIManager uses CreditsPanel, TutorialPanel, HighscorePanel, SettingsPanel, PlayGamePanel, MainMenuPanel.

4.3 Architectural Patterns

4.3.1 MVC Pattern

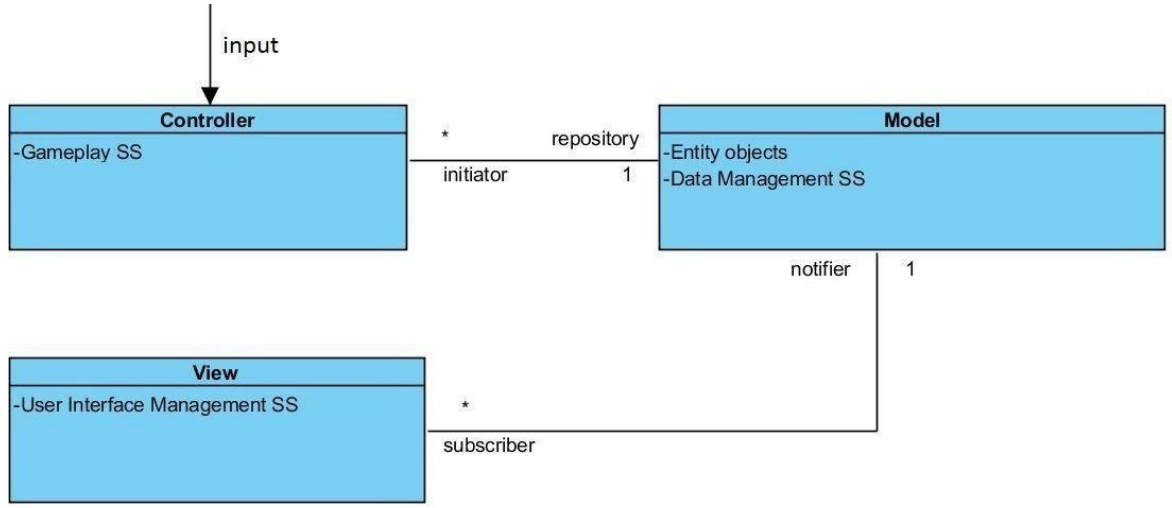

Figure 4: Shows the MVC relation of the Hungry Beast.

Below diagram depicts the sequence of events in MVC. It models how model, view and controller response to a user input during The Hungry Beast.

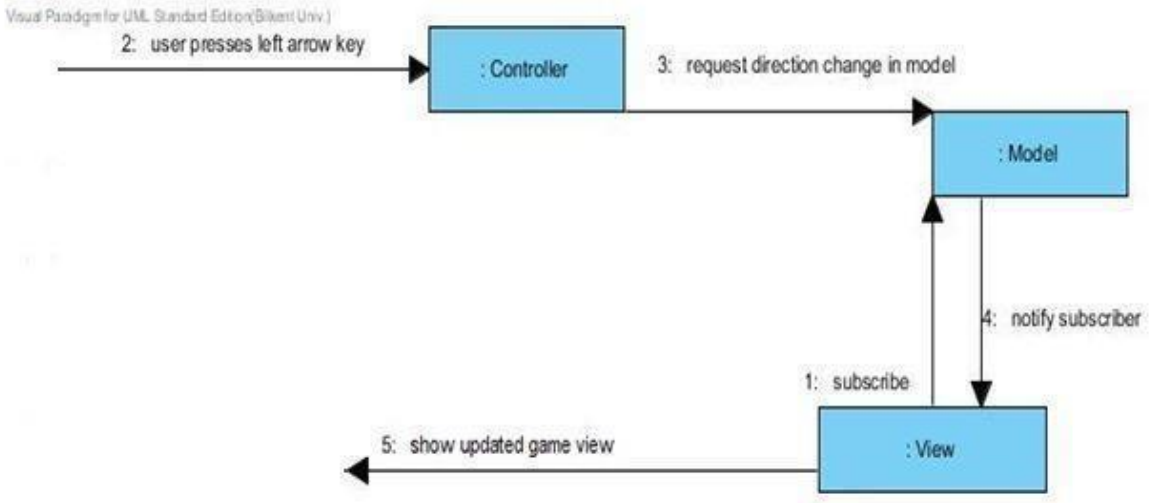

Figure 21: Shows the sequence of events in MVC.

4.3.2 Layers and Partitions

The system is decomposed through three layers and their names are User Interface, Game Control and Game Entity. All of these layers provide related services and they are generated hierarchically. The name of first layer is User Interface. It is the top layer so the top in hierarchy because it is not used by other layers. It provides interaction with the user. In User Interface layer, there are two partitions. One of them is GUIManager class and it provides of management of the screens of the systems. The other partition contains the ScreenManager class which handles the transitions between the screens. (This class addaded to our project)

The second layer is Game Control which is used by User Interface. Its provides to control the flow of the game. Game Control layer consists of three partitions. One of them is Mode Control subsystem in which there are classes responsible from the level events. Another partition is abbot the duty of GamePlay subsystem accounts of player. These two partitions provide services to the middle partition which contains the GameController class.

The bottom layer is Game Entity and it is used by Game Control. It provides to include information about entity objects. This layering has closed architecture so every layer has access to the layer only below it.User interface accesses to Game Control and Game control accesses o Game entity.

4.4 Hardware / Software Mapping

The programming language to implement this project is JAVA since it is a common language that we all know better and also JAVA API is very useful for us, therefore we will use the latest JDK version. The reason why we use the JAVA API is that it allows us to find many classes and methods which we don't know yet that they exist, but very useful for us to implement the project. Also, its compiler specifically shows the errors, even leads the way to correct them. Therefore, it may save our time and effort while implementing the project. Another reason is that most of the computers have already had JAVA on them so the game will run without needing any other software on most of the computers. Shortly, because of these simplicities of JAVA, we prefer to use it to implement our project. The game we develop doesn't need network connection since there is no multiplayer option to play.

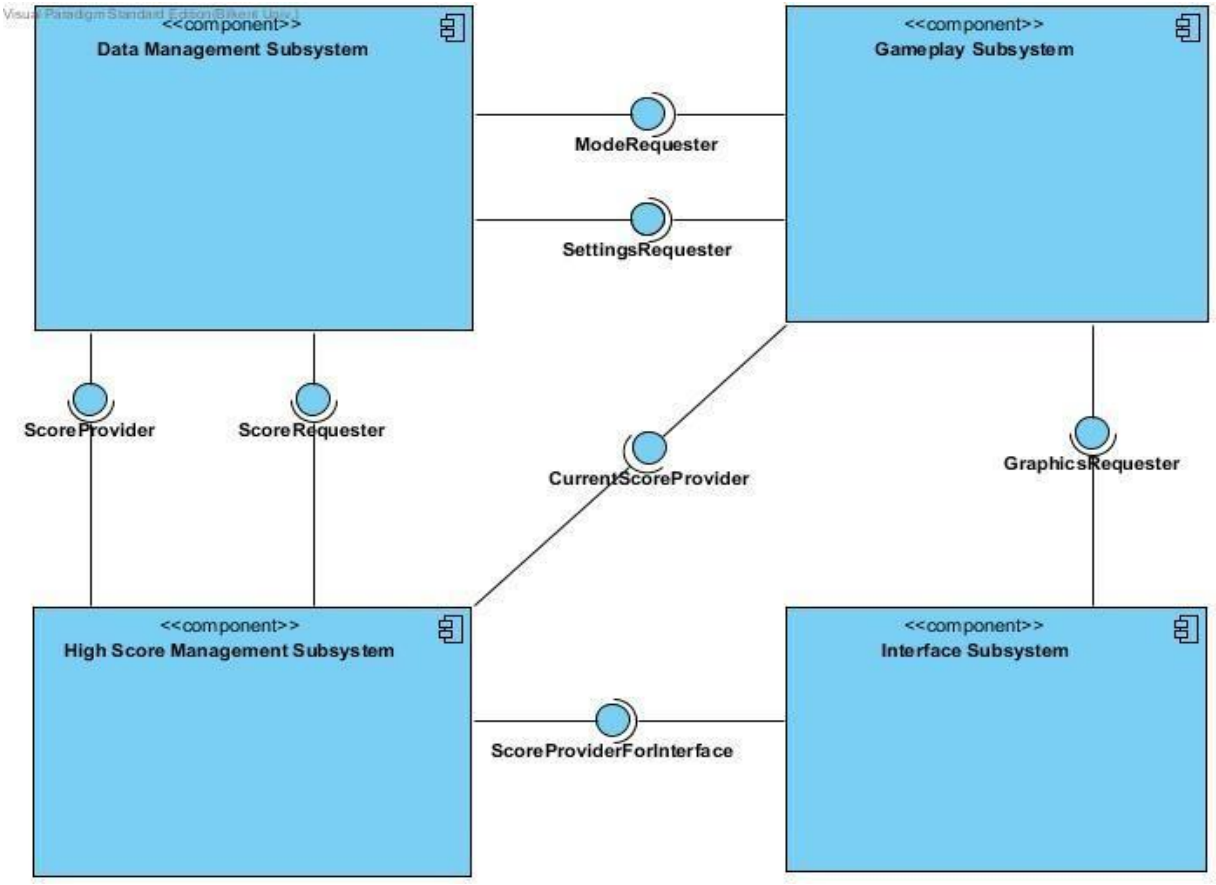

Figure 22: Shows the component diagram.

The dependency relationships between the components (somehow subsystems) are depicted in the component diagram above. In order to complete the hardware/software mapping, dependencies between hardware components also must be clearly stated. Since The Hungry Beast is a single player game (and not an online game), the game does not require the usage of

the internet or some kind of connection with other players. Also, it does not require a database server or any other hardware. The only hardware dependency the application has is the dependency posed by the user in order to control the game by inputting some data from the keyboard and the mouse. Therefore, the hardware mapping is straightforward. It can also be observed from the deployment diagram below.

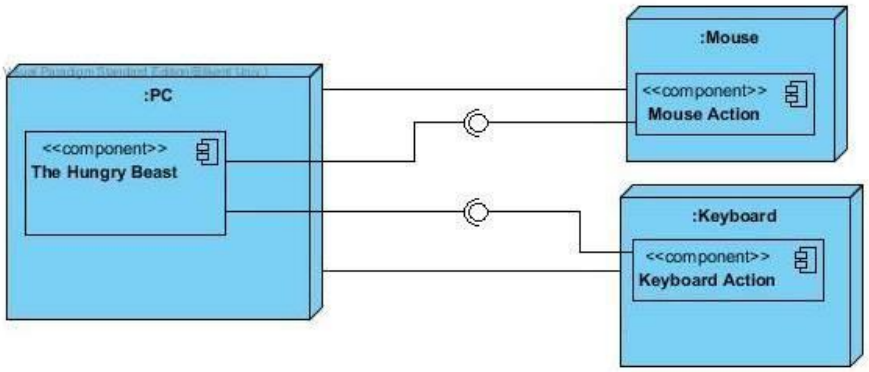

Fig.23: Shows the deployment diagram

4.5 Addressing Key Concerns

Other than the functionalities of the application, there are some concerns that needs to be addressed and some requirements that needs to be met in order to fully satisfy the users of the application. Below, some key concerns that might be important for the users and how we, as the developers of The Hungry Beast, plan to approach to that concerns are explained.

4.5.1 Persistent Data Management

In every system, there are some data that is important for the user, and, hence needs to be saved. The data that needs to be preserved in The Hungry Beast is the high score data. After the user achieves a high score in the game, the application asks the name of the user and then saves that name and score values to later show it on the high score screen. So, there are one writer and multiple readers of this persistent data in the game. Therefore, The Hungry Beast uses a file system to maintain the data. Once the user inputs the name, the name value and the score that corresponds to that user is written into a .txt file. When the user wants to display the high scores by going to the 'High score' screen from the main menu, the application reads the high score data from that .txt file and displays it on the screen.

4.5.2 Access Control and Security

Since The Hungry Beast is a single player game (Also, not an online game), there

is only one user that can access the game during some time period. Therefore, all the users have the access to all of the functionalities of the system. There is no need to be careful about the data security since the only data that is preserved in the game is the high score values. The Hungry Beast does not require membership or does not ask any information from the user except than the name value he/she inputs for high score displaying.

4.5.3 Global Software Control

While deciding on software control, we decided to use one control object to control all of the events since it is easier to apply changes to a single control object. Therefore, we will follow an explicit control pattern. Also, it will be centralized since there will be a single control object, called GameDynamicsEngine. The con of using a centralized design is to have a possible performance bottleneck. In our case, it is not possible because the game we are designing favors simplicity and hence, does not provide many functionalities happening at the same time. In addition to being centralized, the control will also be procedure-driven, meaning that the control will reside in the program code, in other words, the class of GameDynamicsEngine.

4.5.4 Boundary Conditions

A good system needs to be able to foresee the errors and know how to cope with them. Below, three cases during which occurrence of an error is likely, are examined separately.

Initialization:

- When the game starts, the only data that needs to be accessed by the controller is the mode of the game that is chosen by the user. If the user starts without selecting the mode, there will be complications. In order to prevent those complications, we plan to disable the ‘Play Game’ button in the main menu if the user does not specify the game mode he/she wants to play.
- The user interface initializes the background, the health point bar and some food objects during initialization. They will only be created and displayed. The user interface cannot cause an error during initialization of the game.

Termination:

- A single subsystem is not allowed to terminate the application. When the user wishes

to exit the application, all of the subsystems, which are the database, interface and game play subsystems, are terminated concurrently. Therefore, the program is not likely to crash after being terminated by the user.

- To be more specific, the user does not have direct access to the database and the interface subsystems. So, he/she does not have the power to terminate them. The only subsystem that the user can terminate is the game play subsystem by exiting the game. When that happens, both the database and the interface subsystems are notified and then terminated.

Failure:

The system is not likely to fail because the application is not allowed to proceed if the required information is not inputted. The details of this situation is given below:

- The user is only allowed to use the control set he/she specifies from the main menu. Any other button that If he/she does not specify whether to use arrow keys or 'WASD' keys, the default control is set to arrow keys.
- The user cannot start without selecting the preferred game mode.

Only possible failure scenario occurs if the power goes out during the game is being played. When this happens, progress of the user is not recorded since only recorded data of the game is the high scores. So, the user has to start all over.

4.6 Conclusion

For a good analysis, this is the criterion that any other developer can understand this project and explanations without asking us anything about the project. We defined the system interms understood by the customer and user though analyses report so in other words, we provide to bridge the gap between end user and developer. We made requirement elicitation by defining functional and non-functional requirements and then in the analysis part we made dynamic and object modelling. For dynamic modelling, we used sequence diagram for graphical description of the objects participating with use cases and used state chart diagram for depicting the interesting behavior of some objects. For object modelling, we made class identification and finding attributes, methods and associations between classes. Our goal is to present project design report which is totally comprehensible. In order to succeed it, it is needed to evaluate other's perspective, and to check if they would follow our models and interpretations. When it comes to the design part, firstly we restated the nonfunctional requirements of our systems under the name of design goals. After that, we decomposed the system into subsystems to

simplify our jobs as the developers. We decided to decompose the system into three major subsystems, which are Database, Gameplay and the User Interface. We then chose Façade and MVC patterns to model our system as a whole. Finally, we addressed some key concerns, like data management, security, software control and boundary conditions. Thinking about these concerns will help us to develop our application more carefully in the future. We believe that we now are aware of all of the requirements to satisfy the users of the application. In addition, we know have a better understanding of how to implement The Hungry Beast in an efficient way.

5.0 Object Design

This section covers object design of our project.

5.1 Pattern Applications

5.1.1. Facade Design Pattern

Description: Façade Pattern is a structural design pattern. It provides a better interface to the whole system. It helps the understandability of the system and makes the complex codes to understand easier.

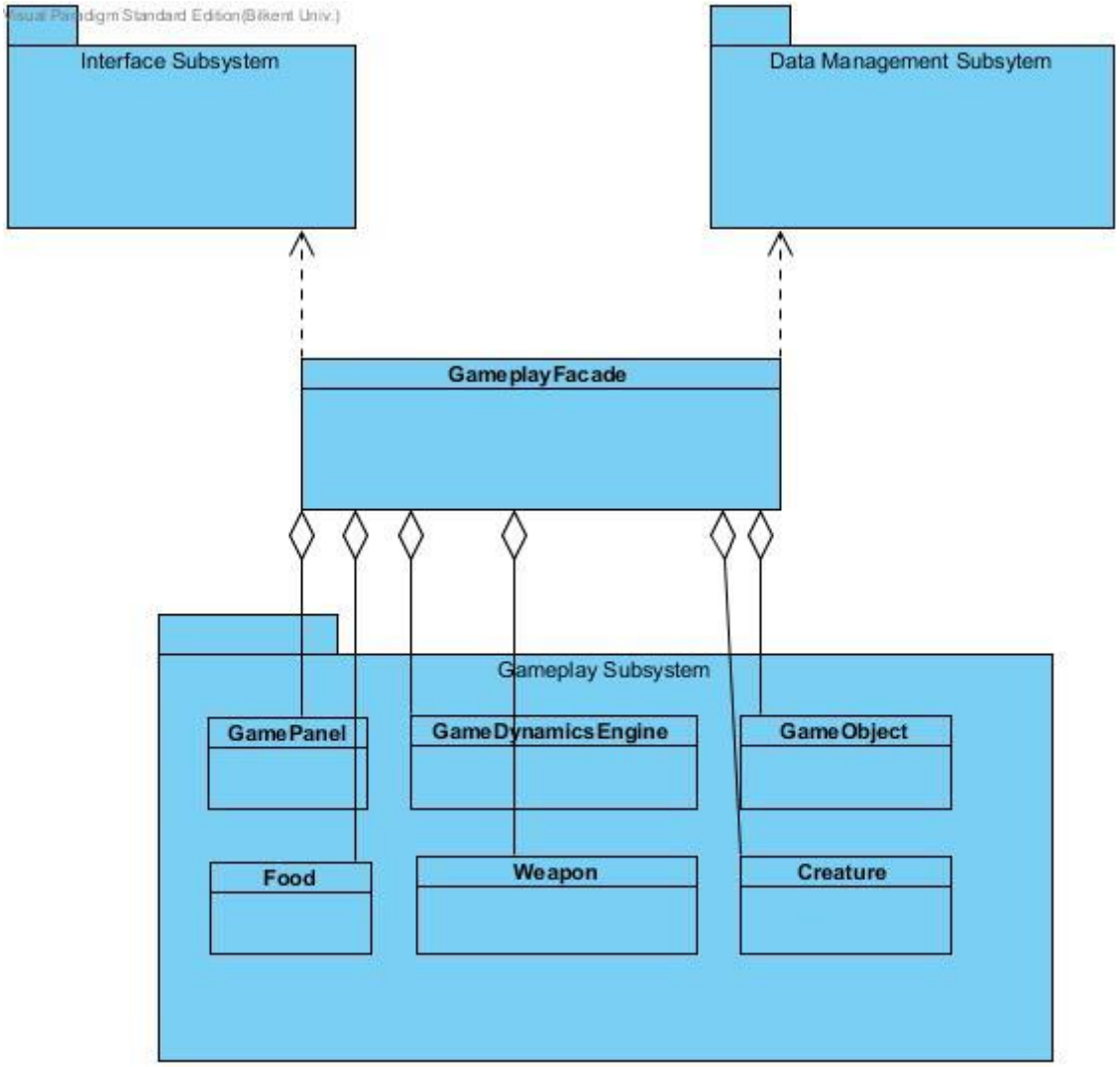

Image 1: Shows Hungry Beast's Façade Design Pattern.

Usage: Weapon, Food and Creature classes, which are abstract, uses Gameplay Subsystem. We used GameplayFacede as the connection between the subsystems. This Façade uses GamePanel, GameDynamicsEngine, GameObject, Food, Weapon and Creature objects. Then makes the relations between interface subsystem and Data Management Subsystem.

5.1.2. Strategy Design Pattern

Description: Strategy Design Pattern is a software design pattern that enables an algorithm to behave differently at the runtime. It defines a family of algorithms, encapsulate them and change their usage at run time.

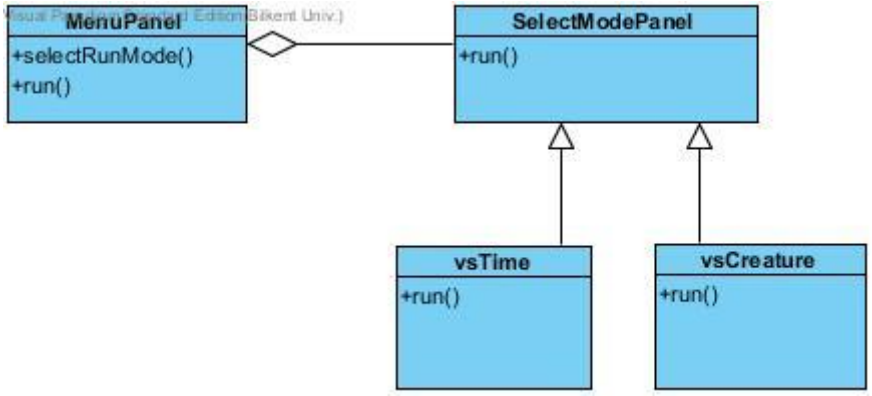

Image 2: Shows Hungry Beast's Strategy Design Pattern.

Usage: We used strategy design pattern in order to decide whether vs Creature game mode is going to run or vs Time mode. In SelectModePanel class, there is a Boolean variable ifTime for making this decision. Program will compile run() method but the method will choose the game mode in run time according to that Boolean variable value.

5.1.3. Composite Design Pattern

Description: Composite Design Pattern is a partitioning design pattern. It describes a group of objects that are going to be treated as a single instance of an object. It composes objects into tree structure to represent whole system.

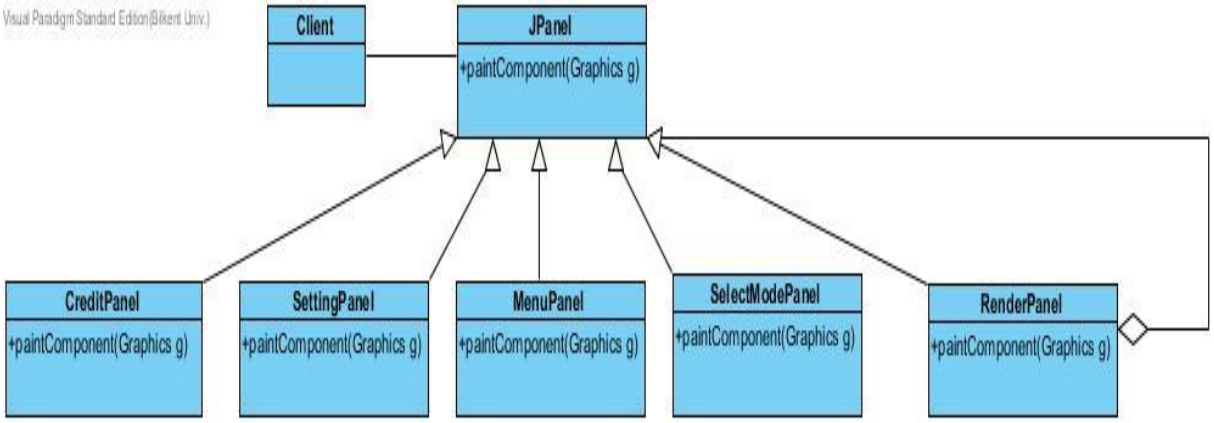

Image 3: Shows Hungry Beast's Composite Design Pattern.

Usage: We used a java class, JPanel, as a parent of the composite design structure. RenderPanel has an aggregation and inheritance relation with the JPanel. The resting panels which are CreditPanel, SettingPanel, MenuPanel, SelectModePanel has only inheritance relation. All classes except from JPanel have paintComponent(Graphics g) function which is derived from JPanel.

5.2 Class Interfaces

Inheritence and Delegation conventions are explained and the interface of every single classes is shown below.

Beast:

Beast
+beastHealth : int
+Beast(String imagePath, float x, float y) +left(): void +right(): void +translateX(float xDif): void +updatePosition(float timePased): void +collided(GameObject g): void +getHealth(): int

Figure 1: Beast Class

Extends: GameObject

This class creates a Beast object with given coordinates and an image. It has a constructor Beast(String imagePath, float x, float y) public property beastHealth(int) and left(void), right(void), translateX(float xDif): void, updatePosition(float timePased): void, collided(GameObject g): void, getHealth(): int as public methods.

Weapon:

Weapon
#power: double
+Weapon(String imagepath, float x, float y) +updatePosition(float timePassed):void

Figure 2: Weapon Class

This class is an **abstract** that enable us to create Weapon objects. It has a protected property power(double) It has a public constructor Weapon(String imagepath, float x, float y) and public method updatePosition(float timePassed):void.

Bullet:

Bullet
+Bullet(x: float, y: float) +collided(GameObject g): void

Extends: Weapon

Figure 3: Bullet Class

This class creates a Bullet object with given coordinates. It has a public constructor Bullet(x: float, y: float) and a public method collided(GameObject g): void.

LaserGun:

LaserGun

+ LaserGun(String imagepath, float x, float y) + collided(GameObject g): void

Extends: Weapon
Figure 4: LaserGun Class
This class creates a LaserGun object with given coordinates.It has a public constructor LaserGun(String imagepath, float x, float y) and a public method collided(GameObject g): void.

Wood:

Wood
+ Wood(String imagepath, float x, float y) + collided(GameObject g): void

Extends: Weapon
Figure 5: Wood class
This class creates a Wood object with given coordinates and an image. It has a public constructor Wood(String imagepath, float x, float y) and a public method collided(GameObject g): void.

Food:

Food
#foodHealth: double #location: Vector2D #enableInc: boolean
+Food(x: float, y: float)

Figure 6: Food Class

This class is an **abstract** class that enables us to create Food. It has 3 protected properties that are foodHealth(double), location(Vector2D), enableInc(boolean) and a public constructor Food(x: float, y: float).

Meat:

Meat
+Meat(x: float, y: float)

Extends: Food
Figure 7: Meat Class
This class is used to create Meat objects by using x and y coordinates. It is derived from Food class. It uses the Food class's properties by calling super(). It also has a unique foodhealth. It has a Meat(x: float, y: float) constructor.

Sugar:

Sugar
+Sugar(x: float, y: float)

Extends: Food
Figure 8: Sugar Class

This class is used to create Sugar objects by using x and y coordinates. It is derived from Food class. It uses the Food class's properties by calling super(). It also has a unique foodhealth. It has a Sugar(x: float, y: float) constructor.

Vegetable:

Vegetable
+Vegetable(x: float, y: float)

Extends: Food

Figure 9: Vegetable Class

This class is used to create Vegetable objects by using x and y coordinates. It is derived from Food class. It uses the Food class's properties by calling super(). It also has a unique foodhealth. It has a Vegetable(x: float, y: float) constructor.

Vector2D:

Vector2D
-x: float -y: float
+Vector2D(x: float, y: float) +getX(): float +getY(): float +scale(value: float): Vector2D +sum(vector: Vector2D): Vector2D +normalize(): void +normalized(): Vector2D +magnitude(): float +dotProduct(vector: Vector2D): float +getProjection(vector: Vector2D): float +substract(vector: Vector2D): Vector2D +distance(vector: Vector2D): float + toString():String +crossProduct(vector: Vector2D): float +setX(x: float): void +setY(y: float): void

Figure 10: Vector2D Class.

This class represents the Vector2D object which will be created for other objects and keep x and y coordinates. With this class, it is possible to use mutator and accessor methods for the objects which is created as Vector2D and it also enable us manage those objects. It has 2 private properties x(float) and y(float). It has a Vector2D(x: float, y: float) constructor and getX(), getY(), scale(value: float), sum(vector: Vector2D), normalize(), normalized(), magnitude(), dotProduct(vector: Vector2D), getProjection(vector: Vector2D), substract(vector: Vector2D), distance(vector: Vector2D), toString(), crossProduct(vector: Vector2D), setX(x: float), setY(y: float) as

public methods.

CreditPanel:

CreditPanel
-serialVersionUID: long -frame: MainFrame -BufferedImage: img
+CreditPanel(frame: MainFrame) +paintComponent(g: Graphics): void +menu(): void

Extends: JPanel

Figure 11: CreditPanel Class

This class is for the GUI and imports many things which are related to the GUI components. It uses many functions which are derived from JPanel and create a unique panel for the credit by arranging location, size and layout. It has also a button which enable the user go back from the CreditPanel with the help of actionListener implementation. Additionally, it is possible to draw an image with the method provided as paintComponent which is derived from the Java's library. It has 3 private properties; serialVersionUID(long), frame(MainFrame), BufferedImage(img). It has a CreditPanel(frame: MainFrame) constructor. It has 2 public methods paintComponent(g: Graphics), menu().

GameDynamicEngine:

GameDynamicsEngine
-objects: ArrayList <GameObject> -rp: RenderPanel -beast: Beast -started: Boolean -frame: MainFrame -settings: SettingPanel -modePanel: SelectModePanel -newGobjPos: Vector2D -goOn: Boolean - shootBlock: Boolean -time: int
+GameDynamicEngine(frame: MainFrame, settings: SettingPanel, modePanel: SelectModePanel) +run(): void +keyPressed(e: Keyevent): void +keyReleased(e: Keyevent): void +keyTyped(e: Keyevent): void +menu(): void +shoot(): void +updatePos(float timePassed): void +createEnemy(float probability): void +checkCollision(): void

Extend: Thread

Implements: KeyListener

Figure 12: GameDynamicsEngine Class

This class is for the GUI and imports many things which are related to the GUI components. It provides the game's dynamic engine as understood by the class name and it consists of a frame and panels. With this class, the program is able to understand the keyEvents with the help of Key's actionListener which is implemented from KeyListener. It also has a button to go to previous menus. It has private properties that are objects(ArrayList <GameObject>), rp(RenderPanel), beast(Beast), started(Boolean), frame(MainFrame), settings(SettingPanel), modePanel(SelectModePanel), newGobjPos(Vector2D), goOn(Boolean), shootBlock(Boolean), time(int). It has a GameDynamicEngine(frame: MainFrame, settings: SettingPanel, modePanel: SelectModePanel) constructor. It has keyPressed(e: Keyevent), keyRealeased(e: Keyevent), keyTyped(e: Keyevent), menu(), shoot(),updatePos(float timePassed), createEnemy(float probability), checkCollision() as public methods.

ImageReader:

ImageReader
+readPNGImage(filePath: String): BufferedImage + TransformColorToTransparency(image: BufferedImage, c1: Color, c2: color): Image + ImageToBufferedImage(image: Image, width: int, height: int): BufferedImage

Figure 13: ImageReader

This class is for the GUI and imports many things which are related to the GUI components and reads the images which are used for the game. With this class, it is possible to draw images which are needed for game. In order not to get any error try-catch clauses is used in this class. It has 3 public methods which are readPNGImage(filePath: String), TransformColorToTransparency(image: BufferedImage, c1: Color, c2: color, ImageToBufferedImage(image: Image, width: int, height: int).

Main:

Main
main(args: String[]): void

Figure 14: Main Class

This class is the core of the game, main panel and frame are created in this class to use.

MainFrame:

MainFrame
-currentPanel: JPanel -gde: GameDynamicsEngine

+MainFrame(currentPanel: JPanel, title: String) +startGame(): void +highscores(): void +tutorial(): void +credits(): void +settings(): void +backtoMenu(): void +play(boolean gameMode): void + gameOver(int score): void

Extends: JFrame

Figure 15: MainFrame Class

This class is for the GUI and imports many things which are related to the GUI components. It is derived from JFrame and uses its components. This class enables user to start game and also keeps the tutorial, credits and settings of the game. It has currentPanel(JPanel) and gde(GameDynamicsEngine) as private properties. It has a MainFrame(currentPanel: JPanel, title: String) constructor and startGame(), highscores(), tutorial(), credits(), settings(), backtoMenu(), play(boolean gameMode, gameOver(int score) as public methods.

MenuPanel:

MenuPanel
-serialVersionUID: long -frame: MainFrame -img: BufferedImage
+MenuPanel() +setFrame(frame: MainFrame): void +paintComponents(g: Graphics): void +startGame(): void +highscoreScreen(): void +tutorialScreen(): void +creditScreen(): void +settingScreen(): void

Extends: JPanel

Figure 16: MenuPanel Class

This class provides MenuPanel which represent the user options to be selected for the game and it is for the GUI and imports many things which are related to the GUI components. It is derived from JPanel and uses its components. In this class, many buttons are created and added to the panel and by implementing actionListeners, it is possible to move the game's screen which are highscorScreen, tutorialScreen, creditScreen and settingScreen. It has serialVersionUID(long), frame(MainFrame), img(BufferedImage) as private properties. It has MenuPanel() constructor. It has setFrame(frame: MainFrame), paintComponents(g: Graphics), startGame(),

highscoreScreen(), tutorialScreen(), creditScreen(), settingScreen() as public methods.

RenderPanel:

RenderPanel
-serialVersionUID: long -objects: ArrayList <GameObject> -backgroundImage: BufferedImage -timeLabel: String
+ RenderPanel() + getGameObjects(): ArrayList + setGameObjects(objects: ArrayList): void + paintComponent(g: Graphics): void + getFormattedTime(int mili): void

Extends: JPanel

Figure 17: RenderPanel Class

This class is for the GUI and imports many things which are related to the GUI components. It is derived from JPanel and uses its components. With this class, it is possible to get objects and even set their properties and with the paintComponents images are drawn. It has serialVersionUID(long), objects(ArrayList <GameObject>), backgroundImage(BufferedImage), timeLabel(String) as private properties. It has RenderPanel(), getGameObjects(): ArrayList, setGameObjects(objects: ArrayList): void, paintComponent(g: Graphics): void, getFormattedTime(int mili): void as public methods.

SelectModePanel:

SelectModePanel
- serialVersionUID: long -frame: MainFrame -img: BufferedImage - ifTime: boolean
+ SelectModePanel(frame: MainFrame) + getIfTime(): boolean + setIfTime(b: boolean) + paintComponent(g: Graphics): void +menu(): void +run(): void

Extends: JPanel

Figure 18: SelectModePanel Class

This class is for the GUI and imports many things which are related to the GUI components. It is derived from JPanel and uses its components. In this class the game is started and the mode is updated according to the selection of the user. It has serialVersionUID(long), frame(MainFrame), img(BufferedImage), ifTime(Boolean) as private properties. It has a constructor which is SelectModePanel(frame: MainFrame) and public methods getIfTime(): boolean, setIfTime(b: boolean), paintComponent(g:

Graphics): void, menu(): void, run(): void.

SettingPanel:

SettingPanel
- serialVersionUID: long -frame: MainFrame -img: BufferedImage -controls: boolean
+settingPanel(frame: MainFrame) +getControls(): boolean +setControls(controls: boolean): Boolean + paintComponent(g: Graphics): void +menu(): void

Extends: JPanel

Figure 19: SettingPanel Class

This class is for the GUI and imports many things which are related to the GUI components. It is derived from JPanel and uses its components. This class is used for the set the specific controls and these changed controls are reflected to the game. It has serialVersionUID(long), frame(MainFrame), img(BufferedImage), controls(Boolean) as private properties. It has settingPanel(frame: MainFrame), getControls(): Boolean, setControls(controls: boolean): Boolean, paintComponent(g: Graphics): void, menu(): void as public methods.

Creature:

Creature
#damage: int
+Creature(String imagePath, float x, float y) +updatePosition(float timePassed): void +collided(GameObject g): void

Extends: GameObject

Figure 20: Creature Class

It creates a creature object. It has a protected property damage(int). It has Creature(String imagePath, float x, float y) as constructor and updatePosition(float timePassed): void, collided(GameObject g): void as public methods.

TutorialPanel:

TutorialPanel
-serialVersionUID: static final long -frame: MainFrame -img: BufferedImage
+ TutorialPanel(MainFrame frame) + paintComponent(Graphics g): void + menu(): void

Extends: GameObject

Figure 21: TutorialPanel Class

It creates the tutorial panel in the game. It has serialVersionUID: static final long, frame: MainFrame, img: BufferedImage as private properties. It has TutorialPanel(MainFrame frame), paintComponent(Graphics g): void, menu(): void as public methods.

GameOverPanel:

GameOverPanel
-serialVersionUID: static final long -frame: MainFrame -img: BufferedImage
+ GameOverPanel(MainFrame frame, int score) + paintComponent(Graphics g): void + menu(): void

Extends: JPanel

Figure 22: GameOverPanel Class

It creates the game over panel in the game. It has serialVersionUID: static final long, frame: MainFrame, img: BufferedImage as private properties. It has GameOverPanel(MainFrame frame, int score), paintComponent(Graphics g): void, menu(): void as public methods.

GameObject:

GameObject
image: BufferedImage # position: Vector2D # collideRadius: float
+ GameObject(String imagePath, float x, float y, float collideRadius) + getImage(): BufferedImage + getPosition(): Vector2D + doesCollide(GameObject g): Boolean + updatePosition(float timePased): abstract void + collided(GameObject g): abstract void

Figure 23: GameObject Class

It creates a GameObject in the game. It has image: BufferedImage, position: Vector2D, collideRadius: float as protected properties. It GameObject(String imagePath, float x, float y, float collideRadius), getImage(): BufferedImage, getPosition(): Vector2D, doesCollide(GameObject g): Boolean, updatePosition(float timePased): abstract void, collided(GameObject g): abstract void as public methods.

DavidDavenport:

DavidDavenport
+ DavidDavenport(float x, float y)

Extends: Creature

Figure 24: DavidDavenport Class

It creates a DavidDavenport object. It has DavidDavenport(float x, float y) as a

constructor.

OkanTekman:

OkanTekman
+ OkanTekman(float x, float y)

Extends: Creature

Figure 25: OkanTekman Class

It creates an OkanTekman object. It has OkanTekman(float x, float y) as a constructor.

OkanTekman:

OkanTekman
+ OkanTekman(float x, float y)

Extends: Creature

Figure 26: WilliamSawyer Class

It creates a WilliamSawyer object. It has WilliamSawyer (float x, float y) as a constructor.

5.3 Specifying Contracts using OCL

Vector2D:

1. **Context** Vector2D:: getX(): float

   post: return x

After the getX() method, x coordinate is returned.

2. **Context** Vector2D:: getY(): float

   post: return y

After the getY() method, y coordinate is returned.

3. **Context** Vector2D:: scale(float value): Vector2D

   post: return new Vector2D(x*value, y*value)

After the scale(float value): method, a new vector is created with x*value and y*value coordinates.

4. **Context** Vector2D:: sum(Vector2D vector): Vector2D

   post: return new Vector2D(vector.getX()+x, vector.getY()+y)

After the sum(Vector2D vector) method, x and y coordinates is summed up with the previous values.

5. **Context** Vector2D:: normalize(): void

   post: x=x/magnitude();

   y=y/magnitude()

After the normalize() method, x and y coordinates are divided by their magnitudes.

6. Context Vector2D:: normalized(): Vector2D

post: return new Vector2D(x/magnitude(),y/magnitude())

After the normalized() method, x and y coordinates are divided by their magnitudes and obtained in 1 Vector2D.

7. Context Vector2D:: magnitude(): float

post: return (float)Math.sqrt(x*x+y*y)

After the magnitude() method, root of x^2 + y^2 is returned.

8. Context Vector2D:: dotProduct(Vector2D vector):float

post: return this.x*vector.getX()+this.y*vector.getY()

After the dotProduct(Vector2D vector) method, dot product of x and y coordinates are obtained.

9. Context Vector2D:: getProjection(Vector2D vector): Vector2D

post: return vector.scale((float)(dotProduct(vector)/(Math.pow(vector.magnitude(), 2))))

After the getProjection (Vector2D vector) method, dot product of a Vector2D is divided by its magnitude square is returned.

10. Context Vector2D:: subtract(Vector2D vector): Vector2D

post: return new Vector2D(this.x-vector.getX(), this.y-vector.getY())

After the subtract (Vector2D vector) method, the coordinates difference of two distinct vectors are obtained.

11. Context Vector2D:: distance(Vector2D vector): float

post: return (float)(Math.sqrt(Math.pow(this.x-vector.getX(),2)+Math.pow(this.y-vector.getY(),2)))

After the distance (Vector2D vector) method, the distance between two point is obtained.

12. Context Vector2D:: crossProduct (Vector2D vector): float

post: return getX()*vector.getY()-vector.getX()*getY()

After the crossProduct (Vector2D vector) method, cross product of a Vector2D coordinates are obtained.

13. Context Vector2D:: setX(float x): void

pre: Vector2D vector = new Vector2D(x,y);

post: this.x=x;

x coordinate of Vector2D is setted to given value.

14. Context Vector2D:: setY(float y): void

pre: Vector2D vector = new Vector2D(x,y);

post: this.y=y;

y coordinate of Vector2D is setted to given value.

Beast:

15. Context Beast:: left(): void

post: position.setX(position.getX()-(float)1.5)

if(position.getX()<0)

position.setX(0);

Beast will go to left if there is a place left to go.

16. Context Beast:: right(): void

post: position.setX(position.getX()+(float)1.5)

if(position.getX()>775)

position.setX(775)

Beast will go to right if there is a place left to go.

17. Context Beast:: translateX(float xDif): void

post: position.setX(position.getX()+xDif)

Beast will move in x coordinate with the xDif parameter.

18. Context Beast:: collided(GameObject g): void

post: if(g instanceof Creature){

beastHealth-=((Creature)g).getDamage();

System.out.println(beastHealth);

}

If a creature will hit to beast, beast's health will decrease by the creature's damage.

19. Context Beast:: getHealth (): int

post: return beastHealth

Returns the beast's health after the function call.

Creature:

20. Context Creature:: updatePosition(float timePassed): void

post: position=position.sum(new Vector2D(0,-0.3f).scale(timePassed))

It updates the creature's position according to the passed time.

GameObject:

21. Context GameObject:: getImage() : BufferedImage

post: return image

Returns to that GameObject's image after the function call.

22. Context GameObject:: getPosition(): Vector2D

post: return position

Returns to that GameObject's position after the function call.

23. Context GameObject:: doesCollide(GameObject g): boolean

post: if(position.distance(g.getPosition())<collideRadius+g.getCollideRadius()){

return true;

}

return false;

Returns true if a collision has occurred between two GameObject's else return false.

Weapon:

24. Context Weapon:: updatePosition(float timePassed) : void

post: position=position.sum(new Vector2D(0,0.5f).scale(timePassed))

Updates the current position of the weapon.

GameDynamicsEngine:

25. Context GameDynamicsEngine:: run () : void

post: rp.repaint()

It runs the game after the function call. It sets the game objects, updates their positions and repaint.

26. Context GameDynamicsEngine:: keyPressed(KeyEvent e): void

post: if (e.getKeyCode()== input)

do smt

It obtains the data that pushed button by the user. If it is 27, game will go to menu. If it is 32, shoot() function is called. If it is 39 or 68, go right. If it is 37 or 65 go left.

27. Context GameDynamicsEngine:: keyReleased(KeyEvent e): void

post: if(e.getKeyCode()==32

shootBlock=false;

If the keyPressed is space it will use the shoot() method.

28. Context GameDynamicsEngine:: shoot(): void

post: objects.add(new Bullet(beast.getPosition().getX() east.getPosition().getY()+50))

After this func

ion call, bullet class will shoot to the upwards at beast's current position.

29. Context GameDynamicsEngine::createEnemy(float probability):void

post: if(probability>Math.random()){

if(Math.random()<0.33f){

objects.add(new DavidDavenport((float)Math.random()*800,500));

}

After this function call, an enemy will be created with random coordinates.

30. Context GameDynamicsEngine:: checkCollision ():void

post: if(objects.get(i).doesCollide)

It will check whether GameObject's collide with each other or not.

6. Conclusions and Lessons Learned

Our project, Hungry Beast is an arcade-style score based game. Our team is tried to develop something different from what's already done in this game genre. In order to achieve that, we focused on the graphical part of the game and we came up with an idea which includes Bilkent instructors. We used our 3 instructors, Okan Tekman, David Davenport and William Sawyer, as invulnerable creatures in the game. By doing that, we aimed the student of Bilkent University and encouraged them to play our game. We decided to use PC as platform and delivered our product only for this platform. Although we did a Greenfield engineering practice, nearly whole platform games have the same rules during the design process. We cloned these processes and implemented our game as creative as possible.

We had 3 big important parts during our project. These are analysis report, and design report and implementation part. In our analysis report we did something new. In CS102 course, we did the simple version of this reporting part but still it is more difficult. We used our knowledge from the previous years and constructed our project roots. We decided on our requirements, scenarios, interface, class diagrams and important state chart and sequence diagrams. By doing the analysis part, we had some solid idea about what are we going to do.

Next, we constructed our design report. Before starting to our design report, we corrected our mistakes from the analysis part and reconsidered our aims again. In design report we stated our goals, subsystem decomposition, architectural patterns, hardware/software mapping, persistent data management, access control and security, global software control and boundary conditions. We detailed our classes and separated them as subsystem in order to make implementation easier.

In the last part of our project, we faced with bigger problems comparing to our reports.

Dividing a project into separate parts is hard. In addition to that, we also need to know our functions and variable names, in order to use them correctly and efficiently. Therefore, we needed a time schedule to arrange this complication. The entities of the project is written first. Then we continued on the gameplay of the project. Lastly, we ended up with the GUI.

To sum up, project was a challenging and educating experience for our group. We learned project development cycle, task assignment, and how to group up individual progress into a group work. At this part, the documentation clearance and design patterns helped us a lot in order to group up our code. We traded some of our design goals from our design report and saw that almost nothing is went as we planned. We faced with unexpected difficulties but our communication as a group helped us to overcome these problems.