Bilkent University

Department of Computer Engineering

Object Oriented Software Engineering

CS 319 Project: The Hungry Beast

Design Report

Eren Erol

Hande Özyazgan

İrem Ergün

Yalın Eren Deliorman

Course Instructor: Özgür Tan

20.11.2015

Contents

[4. Design 3](#_Toc435652778)

[4.1 Design Goals 3](#_Toc435652779)

[4.2 Subsystem Decomposition 4](#_Toc435652780)

[4.3 Architectural Patterns 7](#_Toc435652781)

[4.4 Hardware / Software Mapping 9](#_Toc435652782)

[4.5 Addressing Key Concerns 9](#_Toc435652783)

[4.5.1 Persistent Data Management 10](#_Toc435652784)

[4.5.2 Access Control and Security 10](#_Toc435652785)

[4.5.3 Global Software Control 10](#_Toc435652786)

[4.5.4 Boundary Conditions 11](#_Toc435652787)

# 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.

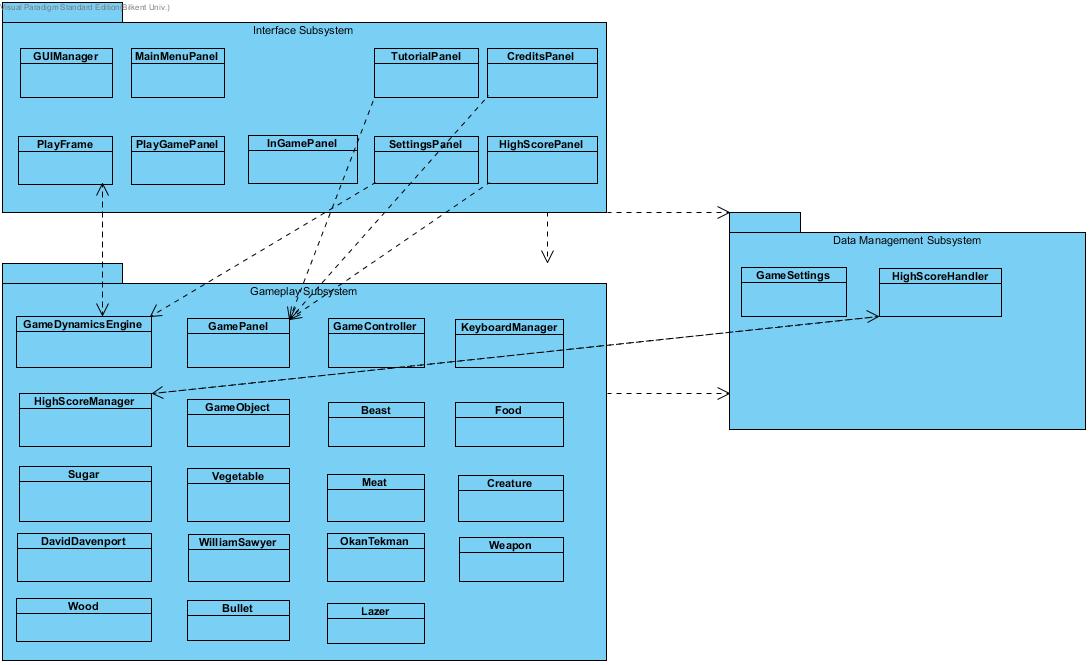
# 4.2 Subsystem Decomposition

The software is composed of three subsystem so 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.

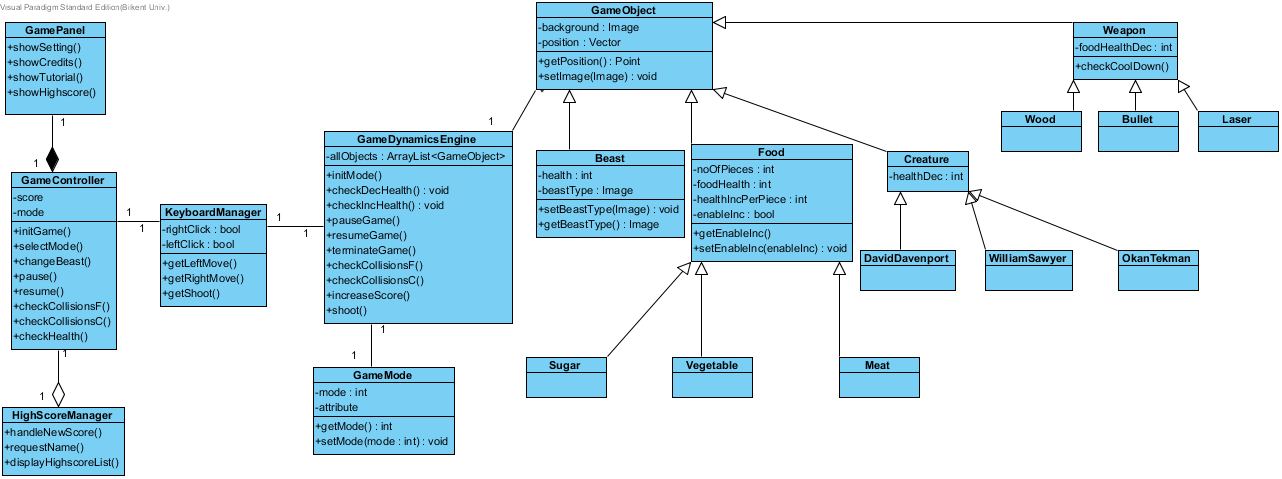
Data Management Subsystem handles storing and accessing game settings, game mode and high score so the interface provided by Data Management Subsystem handles storing settings and retrieves game mode and data of high score which are basic text files.

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 1: Shows the Subsystem relationships.*

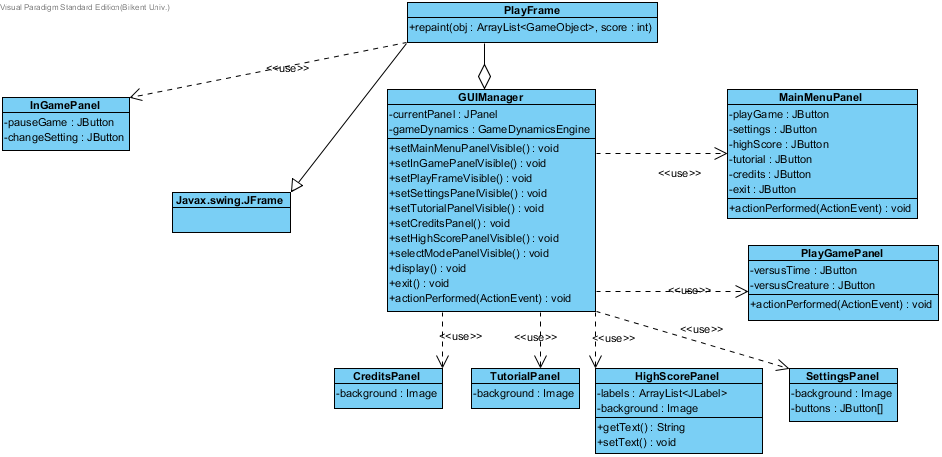
Below you can see the whole GUI independant package of the Hyper Beast. Associations and hiearchy can be seen from the below figure.



*Figure 2: 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.

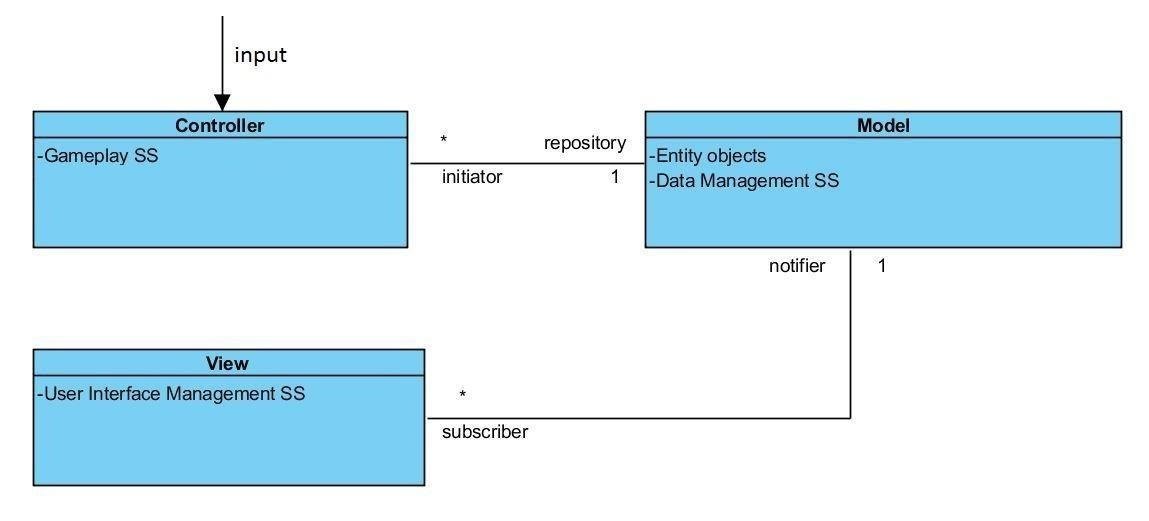


*Figure 3: Shows the GUI package.*

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.

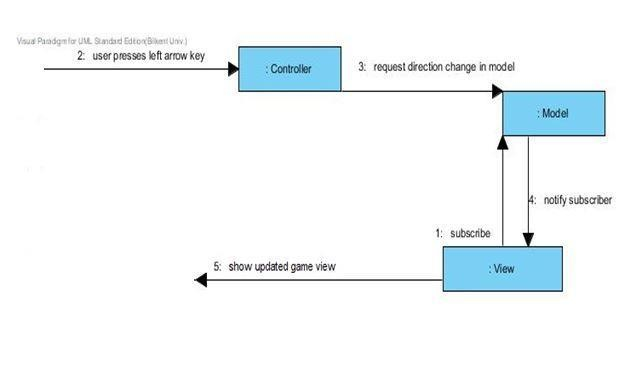
# 4.3 Architectural Patterns

We used 2 architectural design patterns in our design, which are Façade design pattern and MVC (Model-View-Controller) design pattern. We need to divide our project into subsystems to reduce the complexity of the system. We do this by using MVC pattern. Each part of the project is belongs to controller, model or view part.

We used Façade design pattern for making software libraries easier to use and readable also to reduce dependencies of the system. It provides an interface to a long code sample like a library or a subsystem. This action is done by creating an intermediate class which takes the requests and handles them by using the collection of classes that needs to be simplified. While simplifying the classes and subsystems, it also provides reusability, flexibility, extensibility and easy maintenance. We used Façade design pattern in out interface and gameplay subsystem.

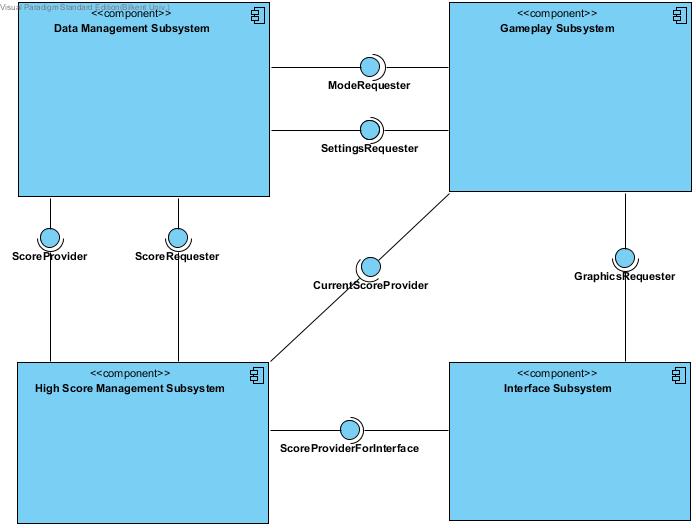
*Figure 4: Shows the MVC relation of the Hungry Beast.*

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 5: Shows the sequence of events in MVC.*

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 6: Shows the component 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.