1. Introduction

1.1 Purpose of the System

Astunc is a bullet hell type of game. Bullet hell is a subgenre of shoot them up games. The main objective is to destroy the enemy entities on screen with the projectiles from user. Usually these games are set on space. We can describe this kind of game as Chicken Invaders with more projectiles on screen that limits the movement of the player. We aim this game to be easy to pick up and play but has a degree of difficulty for people to master.

1.2 Design Goals

1.2.1 **Usability:** We want our game to be easy to pick up and play because of this we don't want any of our interfaces to be too complicated so the user can find anything he/she is searching for.

1.2.2 **Ease of Learning**: Since we want the user to learn the game quickly we didn't put too much complicated systems or mechanics in to the gameplay and the tutorial mode is key to making the game easy to understand.

1.2.3 **Extendibility:** To increase the longevity of the game we want to add new features such as new stages, different player ships, enemies and power ups easily without changing too much of the implementation.

1.2.4 **Portability:** To make our game portable we will implement our system in Java to make use of the platform independency that comes with it.

1.2.5 **Modifiability:** In order to make modifications on the existing parts of the game easy we will minimize the coupling of the subsystems as much as possible, to avoid undesired changes in other parts of the system.

2. Software Architecture

2.1 Overview

This part will be about decomposing our system into subsystems which will help us to maintain the complexity much more easily. In dividing these subsystems, our main concern is about reducing the coupling between subsystems of the Astunc, on the other hand increasing the cohesion of subsystem components. In addition we tried to decompose our system in order to apply MVC(Model View controller) architectural style on our system.

2.2 **Subsystem Decomposition**

Our system got parts that specialized to perform related actions which helps us to have better view on organization of our system. We tried to minimize coupling of the subsystems and maximize their cohesion to increase extendability and modifications that can be done when needed.

2.3 **Hardware/software Mapping**

Astunc will be implemented using Java as programming language. As hardware, Astunc requires a basic keyboard for playing the game, navigating the menus and typing. Since we will implement the project in Java system requirements will be a basic computer with operating system and a java compiler. System will not require any Internet connection to operate and there will not be any complex database.

2.4 **Architectural Style**

Since we are using MVC architectural style, our system is decomposed by three layers, User Interface, Game Data Handler, Game Data. Hierarchy between these layers is as follows. User Interface is above all since it is about displaying and interacting with the user. Game Data Handler comes second, computes the game logic and arranges the data-flow between User Interface and Game Data. Game Data is the last and holds the necessary variables and objects.

In this architectural style, the main approach is classifying the subsystems into three parts, called model, view and controller. By dividing the subsystems into three parts, we isolate the domain knowledge from the user interface by adding a controller part between them. In our system, we grouped our domain objects into game entities layer which constitutes the model of our system. The domain objects of our system is only accessed and controlled by manager classes which are grouped under Game Data Handler layer that constitutes the controller part. We grouped the classes which are responsible for providing the interaction between user and system into User Interface layer, this layer constitutes the View part since it just communicates with the model part via controller part. By this architecture it is achieved that changes on the interfaces do not change the model of the system, therefore it is a good choice to use MVC for games.

2.5 **Persistent Data Management**

Game data will be stored in the client hard disk drive, we will not use any database since the data we use in the game needs to be accessed in real-time. All the necessary files will be loaded to memory and accessed as needed.

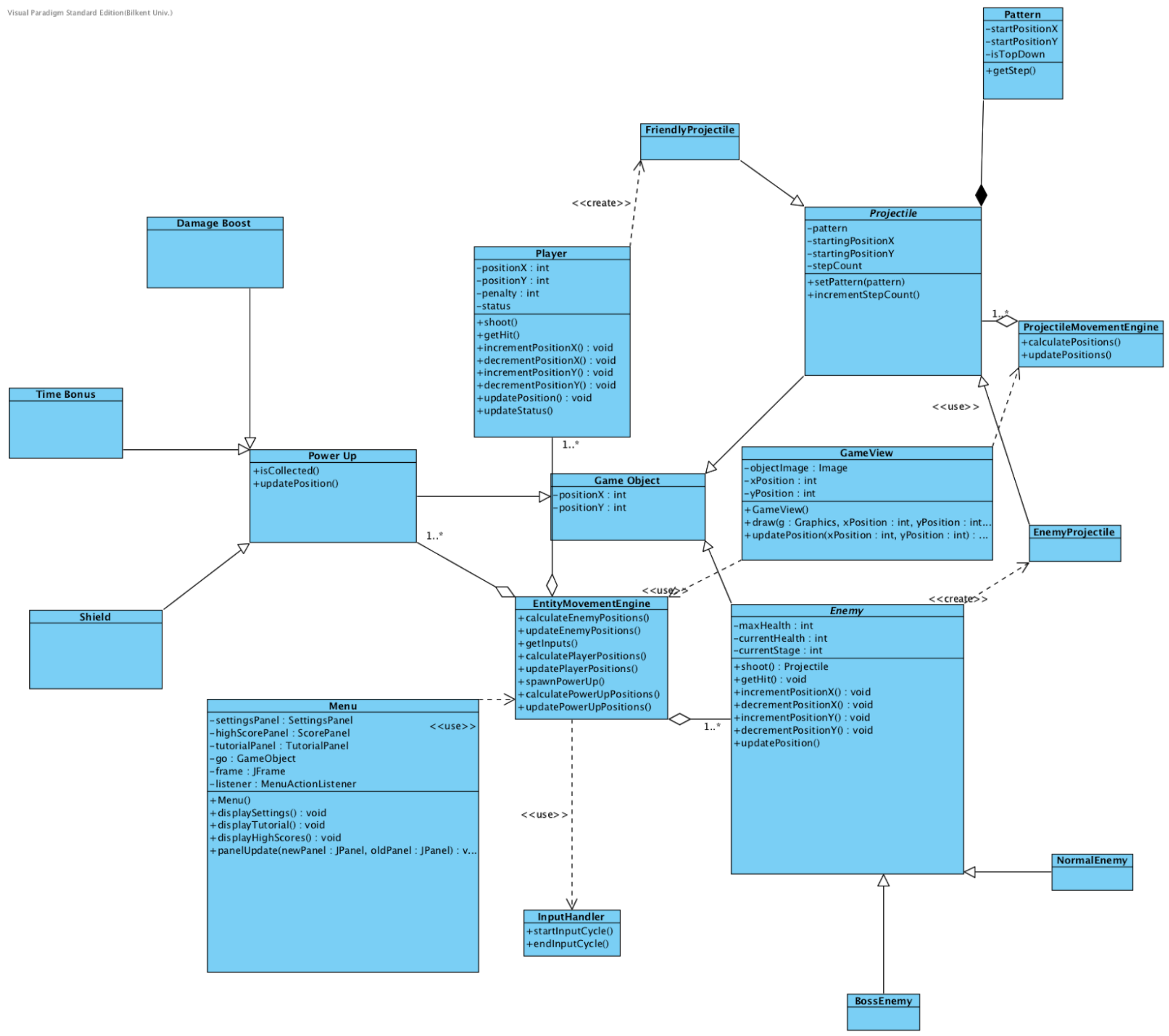
2.6 **Access Control and Security**

Astunc will not require any kind of internet connection. And since there isn’t any kind of database to hold credentials as well as no need for any authentication anyone who runs the program will be able to play the game. And since there are no kinds of user profile there won’t be any kind of security problems

2.7 **Boundary Conditions**

Astunc will not require any kind of install. Astunc can be closed using the Quit button in the main menu if the player wants to quit during gameplay player must first pause the game then return to main menu in order to quit. When player finishes all of stages game will end and player will be sent to the display high scores screen and this screen will update to show current player’s name if a record is broken. Game will give an error if the file is corrupted.

3. Subsystem Services



Menu

Attributes:

Private JFrame frame: Frame for displaying the visual context of Astunc.
Private MenuActionListener listener: A listener which allows to gain inputs from user to   use them for computations and navigations.
Private SettingsPanel settingsPanel: A JPanel instance to display settings menu on screen.

- SettingsPanel class constructs this property with its graphical user interface components like labels and buttons.

Private TutorialPanel tutorialPanel: A JPanel instance to display highscores on the screen.

- TutorialPanel class constructs this property with an GameObject(This class also in model part) and in this panel a mini game will be constructed to user.

Private ScorePanel highScorePanel: A JPanel instance to display highscores on the screen.

- ScorePanel class constructs this property with its graphical user interface components like labels and buttons.

Private GameObject go: This property is an GameObject(This class is also in the model part) instance and it is created when the Play Game choice from the menu is selected.

Constructor:
Public Menu: Initializes all properties of Menu Class

Methods:
Public void displaySettings(): This method puts *settingsPanel* to the screen with the help of *panelUpdate* method.
Public void displayTutorial(): This method puts *tutorialPanel* to the screen with the help of *panelUpdate* method.
Public void displayHighScores(): This method puts *highScorePanel* to the screen with the help of *panelUpdate* method.
Public void panelUpdate(newPanel, oldPanel): This method changes panels according to users choices.

GameView

Attributes
Private Image objectImage: This property have the images for game objects like spaceships and bullets etc…
Private int xPosition: This property holds the x-position of objects which will be drawn**.**
Private int yPosition: This property holds the y-position of objects which will be drawn.

Constructor
Public GameView(): Initializes the properties of the class.

Methods
Public void draw(g, xPosition, yPosition): This methods draws the object with the Image in class and positions.
Public void updatePosition(xPosition, yPosition): This method updates the positions of drawn object for redrawing.

Pattern

This class is the representation of the bullet patterns that enemies and boss enemies fire as. Because of this it has starting coordinates and a method to fetch the steps of the pattern.

Attributes:

1- startPositionX: X coordinate of the starting position for the pattern.

2- startPositionY: Y coordinate of the starting position for the pattern.

3- isTopDown: Boolean: Boolean variable that shows whether the pattern is top down.

Methods:

1- getStep(): Fetches the steps of the pattern.

Projectile

This class represents the projectiles that are fired in the game. This is a parent class that holds attributes that are common to both friendly and enemy projectiles.

Attributes:

1- Pattern: pattern of projectiles that are being fired (pattern class explained above)

2- startingPositionX: X coordinate of starting position for the projectile.

3- startingPositionY: Y coordinate of starting position for the projectile.

4- stepCount: Holds the information about which step of the pattern projectile is in.

Methods:

1- setPattern(pattern): Allows setting of the pattern that projectile is being fired

2- incrementStepCount(): Simply increments the stepCount after a step of the pattern is fired.

Enemy Projectile

Child class of the projectile that represents enemy projectiles which are the projectiles that normal and boss enemies fire. (Normal and boss enemies are explained below.)

Friendly Projectile

Child class of the projectile that represents friendly projectiles. Currently limited to players projectiles.

Player

Class that represents the player's character in the game. Has position and the ability to shoot and move.

Attributes:

1- positionX: X coordinate of the player's position.

2- positionY: Y coordinate of the player’s position.

3- Penalty: Shows the amount of time penalty the player currently has on it.

4- Status: Shows whether player has any power ups active. (Power-ups are explained below)

Methods:

1- Shoot(): Creates friendly projectiles according to players inputs.

2- getHit(): Increases the time penalty of the player every time it gets hit.

3- incrementPositionX(): Increments positionX according to players inputs.

4- decrementPositionX(): Decrements positionX according to players inputs.

5- incrementPositionY(): Increments positionY according to players inputs.

6- decrementPositionY(): Decrements positionY according to players inputs.

7- updatePosition(): Updates players position on the screen.

8- updateStatus(): Updates players status as a power up is collected.

Enemy

Class that represents the enemies in the game. Has position and shoots projectiles as patterns. Parent class for normal and boss enemies. Child class of game object.

Attributes:

1- maxHealth: Maximum amount of health an enemy can have.

2- currentHealth: Current amount of health and enemy has.

3- currentStage: Shows the stage of an enemy. (Enemies have stages as their current health reduces they start to use different patterns of projectiles.)

Methods:

1- shoot(): Creates enemy projectiles that fly in a pattern.

2- getHit(): Reduces the enemies current health as it gets hit.

3- incrementPositionX(): Increments the variable positionX that this class inherits from its parent.

4- incrementPositionY(): Increments the variable positionY that this class inherit from its parent.

5- decrementPositionX(): Decrements the variable positionX that this class inherits from its parent.

6- decremrntPositionY(): Decrements the variable positionY that this class inherits from its parent.

7- updatePosition(): Updates the enemy position on screen.

Normal Enemy

Class that represents the normal enemies that spawn through the level usually have 2 health and no stages. Child class of Enemy.

Boss Enemy

Class that represents the boss enemies that spawn at the end of a level usually have higher health than normal enemies and have stages. Child class of Enemy.

Power Up

Class that represent the power ups that appear on the level that player can collect to gain different bonuses according to the power up's type. Parent class of all types of power ups. Child class of game object.

Methods:

1- isCollected(): Calls the updatePosition() to remove the power up from display as it is collected.

2- updatePosition(): Updates the position of the power up.

Time Bonus

Class that represent one of the possible power ups that appear in a level. Reduces the current penalty of the player if player has 0 penalty gives the player negative penalty. Child class of power up.

Shield

Class that represent one of the possible power ups. Gives the player a shield that will negate the first hit that the player takes.

Damage Boost

Class that represent one of the possible power ups. Gives the player a damage boost that allow player's projectiles to deal 1 more damage to all enemies.

Game Object

Base class for objects in the game that has positions in the screen. Parent class of projectile power up and enemy.

Attributes:

1- positionX: X coordinate of the object.

2- positionY: Y coordinate of the object.