
SEROTOPE SOFTWARE DESIGN DOCUMENT

BENEDICT HOBART LEANNE CAPEWELL JORDAN BURDETT
BHOBART LEANNEEC JBURDETT

TABLE OF CONTENTS

- 1. Introduction**
 - 1.1.Purpose
 - 1.2.Scope
 - 1.3.Overview
 - 1.4.Definitions and Acronyms
- 2. Use Cases**
 - 2.1.Actors
 - 2.2.List of Use Cases
 - 2.3.Use Case Diagrams
 - 2.4.Use Cases
- 3. Analysis Overview**
 - 3.1.Domain Model
 - 3.2.Component Diagram
 - 3.3.Subsystems
 - 3.4.Subsystem Interfaces
- 4. Design Overview**
 - 4.1.Classes
 - 4.2.Class Dictionary
- 5. Dynamic Model**
 - 5.1.Sequence Diagrams
 - 5.2.Activity Diagrams
 - 5.3.State Diagrams
- 6. Data Design**
 - 6.1.Data Description
 - 6.2.Data Dictionary
- 7. Human Interface Design**
 - 7.1.Overview of Human Interface
 - 7.2.Screen Images
 - 7.3.Screen Objectives and Actions
- 8. Appendices**

1. INTRODUCTION

This software design document describes the architecture and system design of Serotope to aid in its development by detailing how it should be built. Within this document are narratives and semi-formal notation detailing its design including use case models, sequence diagrams, domain models and class diagrams.

1.1. PURPOSE

The purpose of this design document is too provide the developers with an astute understanding of what is to be done in response to the objectives of our stakeholders it does this by providing a description of the software system to be built.

1.2. SCOPE

This software design document details the design for an educational game ‘Serotope’ that provides high school students with a means to understand the principles of Mendelian inheritance in an intuitive sense. It is not to be used as the sole method of education for Mendelian inheritance.

1.3. OVERVIEW

Serotope is a multi-directional shooter similar to the game ‘Asteroids’, the user controls a creature by moving it around a two-dimensional world inhabited by other creatures. The objective of the game is to get the high-score by surviving the longest amount of time, the user does this by moving, shooting and ‘reproducing’ with other creatures to produce offspring and thus survive for another generation.

1.4. DEFINITIONS AND ACRONYMS

- **JBOX2d** – *a Java port of the physics engine Box2d initially designed for C++*
- **Slick** – *a java based game engine built on the LWJGL framework*
- **Medal** – *an un-lockable individual achievement for the game*

2. USE CASES

2.1. ACTORS

2.1.1 USER

The User is the person who interacts with the game. This is most likely a student playing the game for education, but this abstraction represents all users, they will perform similar actions.

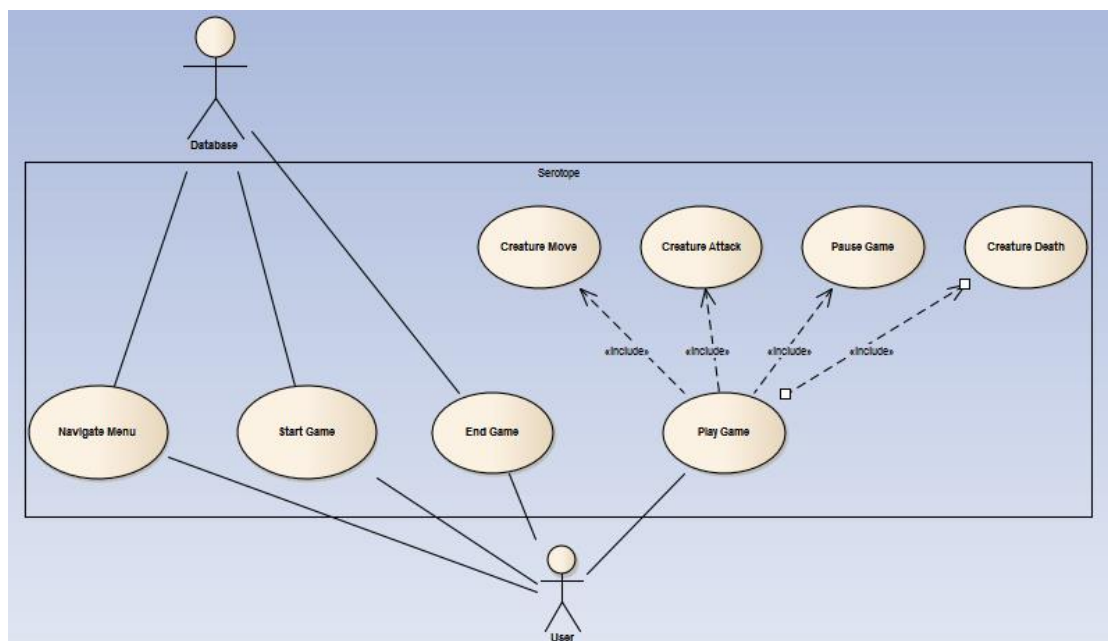
2.1.2 DATABASE

The database is the local data saved on the User's computer. This is how the system under design manages high scores and achievements, with possible implementation of save-states.

2.2. LIST OF USE CASES

- 2.2.1 Navigate Menu
- 2.2.2 Start Game
- 2.2.3 Play Game
- 2.2.4 Move creature
- 2.2.5 Creature attack
- 2.2.6 Creature Death
- 2.2.7 Pick up DNA
- 2.2.8 Pause Game
- 2.2.9 End game

2.3. USE CASE DIAGRAM



2.4. USE CASES

2.4.1 NAVIGATE MENU

Actors

User
Database

Stakeholders and their needs

- User
- needs easy and quick access to their desired screen
 - needs to know how to play the game
 - needs to view achievements
 - needs to change settings
 - needs to enter new game
- Database
- needs access to leaderboard and scores
 - leaderboard information needs to be up to date

Preconditions

User has loaded the game and is at the main menu splash screen.

Postconditions

None

Trigger

User presses any key

Basic flow

1. Main menus screen is displayed
2. User selects Play Game
3. Exits Menu screen.
4. Enters Gameplay screen.

Extensions

- 2a. User selects 'Achievements'
- 2a1. Local storage checked for game data
 - 2a1a. Data is present
 - 2a1a1. Load data
 - 2a1a2. Display Achievement screen with scores from data
 - 2a1b. Data isn't present
 - 2a1b1. Create local store
 - 2a1b2. Display Achievement screen with no Achievements
- 2b. User is not at Main Menu screen and wishes to return to it
- 2ba. User is in Gameplay screen
 - 2ba1. User opens Pause Menu
 - 2ba2. User selects Main Menu option
 - 2ba3. Ends game and exits Gameplay screen
 - 2ba4. Opens Main Menu screen
 - 2bb. User is in a subsection of the Menu screen
 - 2bb1. User selects Go Back option

- 2bb2. Main Menu screen is displayed
- 2c. User selects 'How To' option
 - 2c1. Tutorial screen is loaded at page 1
 - 2c2. User navigates tutorial screen using onscreen "Previous Page" and "Next Page" arrows
 - 2c2a. User is at first page of tutorial
 - 2c2a1. Previous Page arrow is transparent
 - 2c2a2. User selects Previous Page arrow
 - 2c2a2a. Nothing is executed
 - 2c2a3. User selects Next Page arrow
 - 2c2a3a. Next page of tutorial is displayed
 - 2c2a3b. Previous page arrow is no longer transparent
 - 2c2b. User is at last page of tutorial
 - 2c2b1. Next page arrow is transparent
 - 2c2b2. User selects Previous Page arrow
 - 2c2b2a. Previous page of tutorial is displayed
 - 2c2b2b. Next Page arrow is no longer transparent
 - 2c2b3. User selects Next Page arrow
 - 2c2b3a. Nothing is executed
 - 2c2. User navigates tutorial screen using onscreen "Previous Page" and "Next Page" arrows
- 2d. User selects 'Quit Program' option
 - 2d1. Program quits to desktop
- 2e. User selects the cog option representing settings
 - 2e1. Settings menu is loaded onscreen
 - 2e2. User makes changes to the game's settings

2.4.2. START GAME

Actors

User
Database

Stakeholders and their needs

User – wants to start the game so they can play

Preconditions

User has selected "play" on the main menu screen

Postconditions

The game is running the Play Game state

Trigger

User has selected "play" on the main menu

Basic flow

1. A black loading screen is displayed and control is taken away from the User
2. Assets are loaded
3. Once assets have loaded the game screen fades in and control is passed back to the User

4. After game has loaded, the 'Play Game' use case executes

Extensions

- 2a. Assets not present
 - 2a1 Fatal error. Tell user to reload the game
- 2b Game is out of focus
 - 2b1 Pause game

2.4.3. PLAY GAME

Actors

User

Stakeholders and their needs

User – wants to play the game

Preconditions

The game has been loaded

Postconditions

The game is being displayed

Trigger

User has started a game.

Basic flow

1. The game screen is centered on the Creature the user has control of.
2. The user gives input to either shoot move or pause the game.
3. The game loop proceeds until the game 'ends'.

Extensions

- 1a. The user presses a movement key
 - 1a1. The use case "move creature" is executed.
- 1b. The user presses a shooting key
 - 1b1. The use case "creature attack" is executed.
- 2a. The user presses the pause button
 - 2a1. The use case "pause game" is executed.

2.4.4. MOVE CREATURE

Actors

User

Stakeholders and their needs

User – wants to move their creature on the screen

Preconditions

User has started a game and it is currently being displayed on the screen

Postconditions

The game is being displayed.

Trigger

User presses one of the move buttons on the keyboard (default is W, A, S, D).

Basic flow

1. User's creature moves in the direction indicated by the user's input (up, down left or right).
2. Movement speed and acceleration is based on the creatures 'traits'.

Extensions

- 1a. User pressed more than one button
 - 1a1. The user's creature moves in a direction that is the average of all the pressed directions.
- 1b Another creature is in the area the user is trying to move to
 - 1b1. The user's creature is deflected around the other creature.

2.4.5. CREATURE ATTACK

Actors

User

Stakeholders and their needs

User – Wants to attack other creatures by shooting at them

Preconditions

User has started a game and it is currently being displayed on the screen

Postconditions

None

Trigger

User presses one of the shoot buttons on the keyboard (default is arrow keys)

Basic flow

1. User presses the shoot button on the keyboard
2. Bullets are displayed on the screen, moving in the chosen direction, away from the creature. The number of bullets and bullet speed is determined based on the creature's 'traits'
3. The bullet is removed from the game when it interacts with another creature or moves too far off the screen.

Extensions

- 4a Bullet collides with another creature
 - 4a1 That creature takes damage, reducing its hit points based on the strength of the bullet.

2.4.6. CREATURE DEATH

Actors

User

Stakeholders and their needs

User – Wants to kill other creatures and not die

Preconditions

User has started a game and it is currently being displayed on the screen

Postconditions

None

Trigger

A creature has health less than or equal to zero.

Basic flow

1. Creature death animation is displayed.
2. Creature is de-spawned.
3. The creature's 'DNA' is left in the position where the creature died.

Extensions

- 2a Creature that died was controlled by the user.
- 2a1 The game ends (refer to 'End Game' use case)

2.4.7. PICK UP DNA

Actors

User

Stakeholders and their needs

User – Wants to evolve their creature into one with better traits.

Preconditions

User has started a game and it is currently being displayed on the screen

Postconditions

None

Trigger

User's creature moves to the location of a piece of DNA

Basic flow

1. User's creature is displayed at same location as the piece of 'DNA'.
2. The 'DNA' is no longer displayed.
3. User is no longer in control of their creature.
4. Old User controlled creature is taken over by AI.
5. A new creature spawns, and the User is in control of it.

Extensions

None

2.4.8. PAUSE GAME

Actors

User

Stakeholders and their needs

User – Wants to pause the game so they can resume it later.

Preconditions

User has started a game and it is currently being displayed on the screen

Postconditions

The game may be in the paused state.

Trigger

User presses the pause button

Basic flow

1. The game state is paused.
2. All creatures and projectiles stop moving.
3. Timers stop
4. A pause overlay screen is displayed.
5. User chooses to resume game.
6. The game state is restored to exactly the same as it was when it was paused.

Extensions

- 5a. User selects "end game" option
5a1. The game ends, and the main menu is displayed

2.4.9. END GAME

Actors

User
Database

Stakeholders and their needs

User – wants to stop playing the game

Preconditions

User has started a game and it is currently being displayed on the screen

Postconditions

High score screen is displayed.

Trigger

User's creature's health is less than or equal to zero.

Basic flow

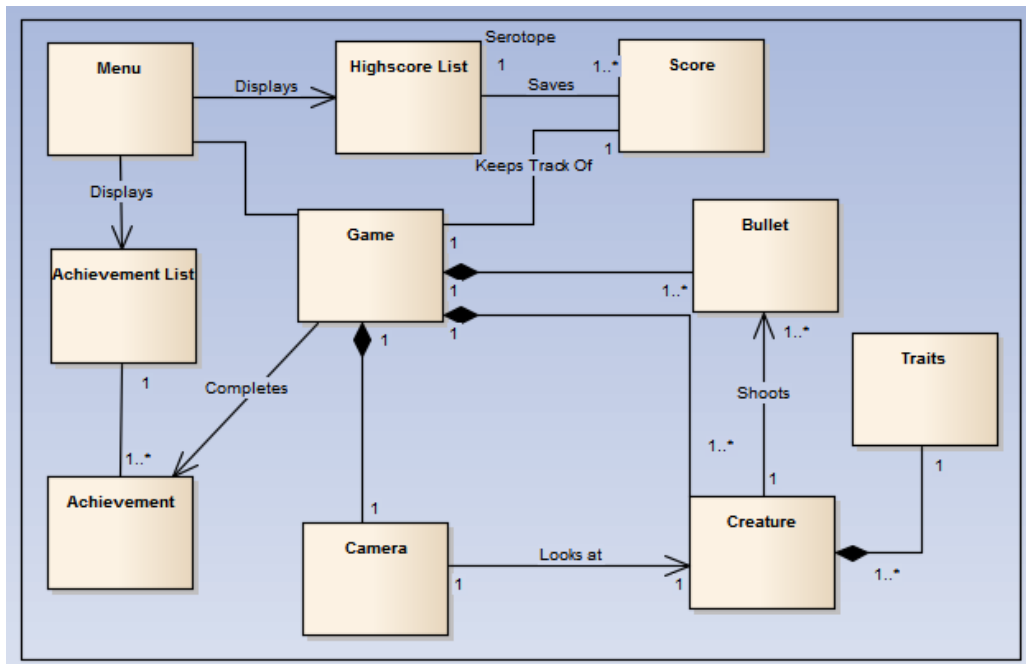
1. Creature death animation is displayed
2. 'Game Over' appears on the screen
3. User's score appears, with "replay" and "return to main menu" options

Extensions

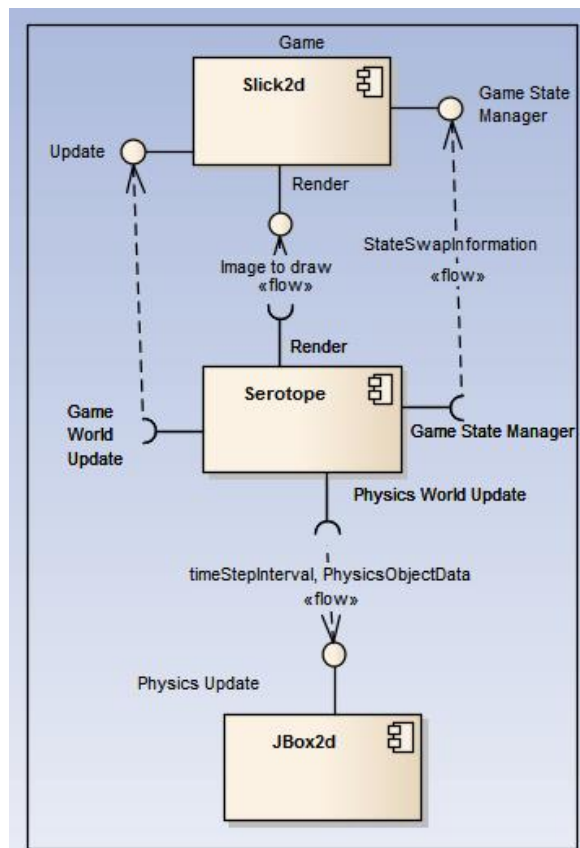
None

3. ANALYSIS OVERVIEW

3.1. DOMAIN MODEL



3.2 COMPONENT DIAGRAM



3.3. SUBSYSTEMS

3.3.1. SEROTOPE

Serotope is the system under design.

3.3.2. SLICK2D

Slick2d is built on top of LWJGL, it manages the frame rate of the game, the different game-states, the update and render mechanisms, and low-level frameworks for playing audio, drawing graphics and detecting user input.

3.3.3. JBOX2D

JBOX2d is a port of Box2d for C++, which provides a library to create a physics world and populate it with bodies. These bodies interact via forces and impulses, which will be detailed in section 6.1.

3.4 SUBSYSTEM INTERFACES

Slick provides Serotope with

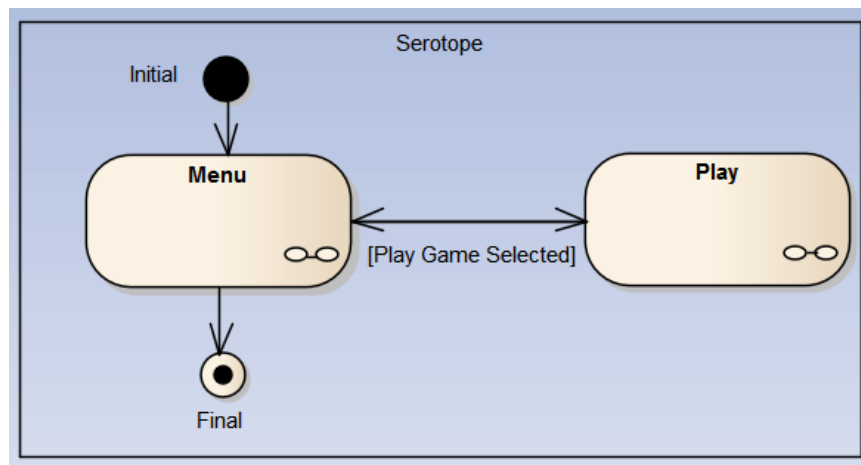
- A game state manager: to swap between the Game and Menu States.
- An update function: creates a way for Serotope to update at approximately regular intervals, as well as provide the time between those update intervals, used for in game calculations.
- A render function: creates a graphics object that will be drawn to the screen and, in the optimal situation, be drawn to the screen at regular intervals.
- High-level frameworks: to play audio and get user input.

JBox2d provides Serotope with

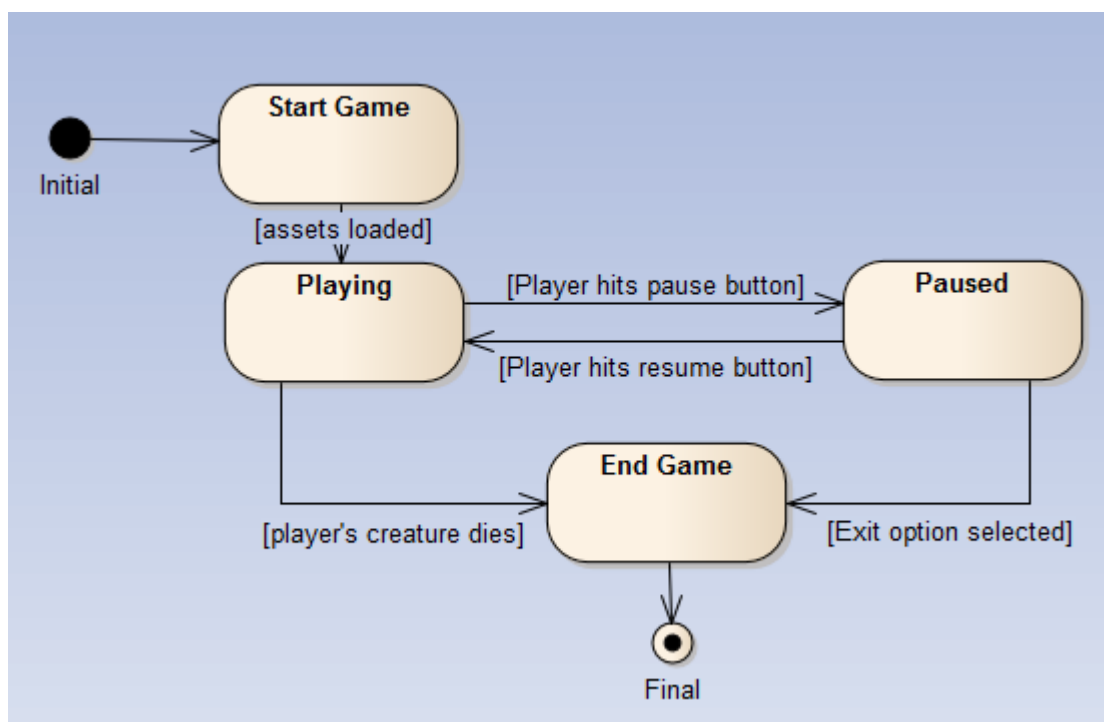
- Collision detection
- Physics simulation
 - Velocity, Momentum, Force, Impulse, Mass, Angular Momentum and Angular Velocity based calculations.
- Low-Level query mechanism to the physics world, including Ray-traces and Axis-aligned bounding box queries.

3.5 ANALYSIS STATE DIAGRAMS

3.5.1 GAME MENU DIAGRAM



3.5.2 GAMEPLAY STATE DIAGRAM



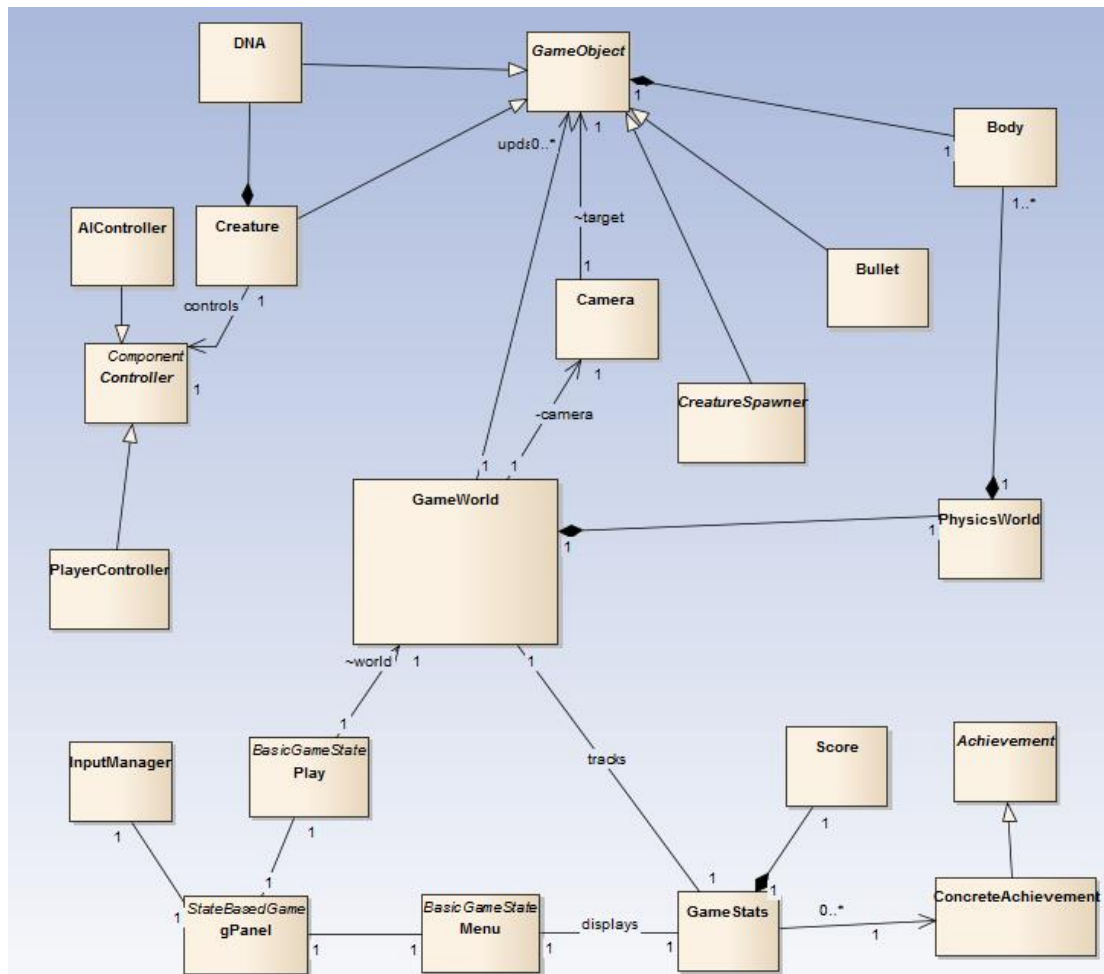
4. DESIGN OVERVIEW

4.1. CLASS DIAGRAM

Due to size and readability constraints of the document, the complete class diagram is included as an .eap file in the submission, and in the appendix.

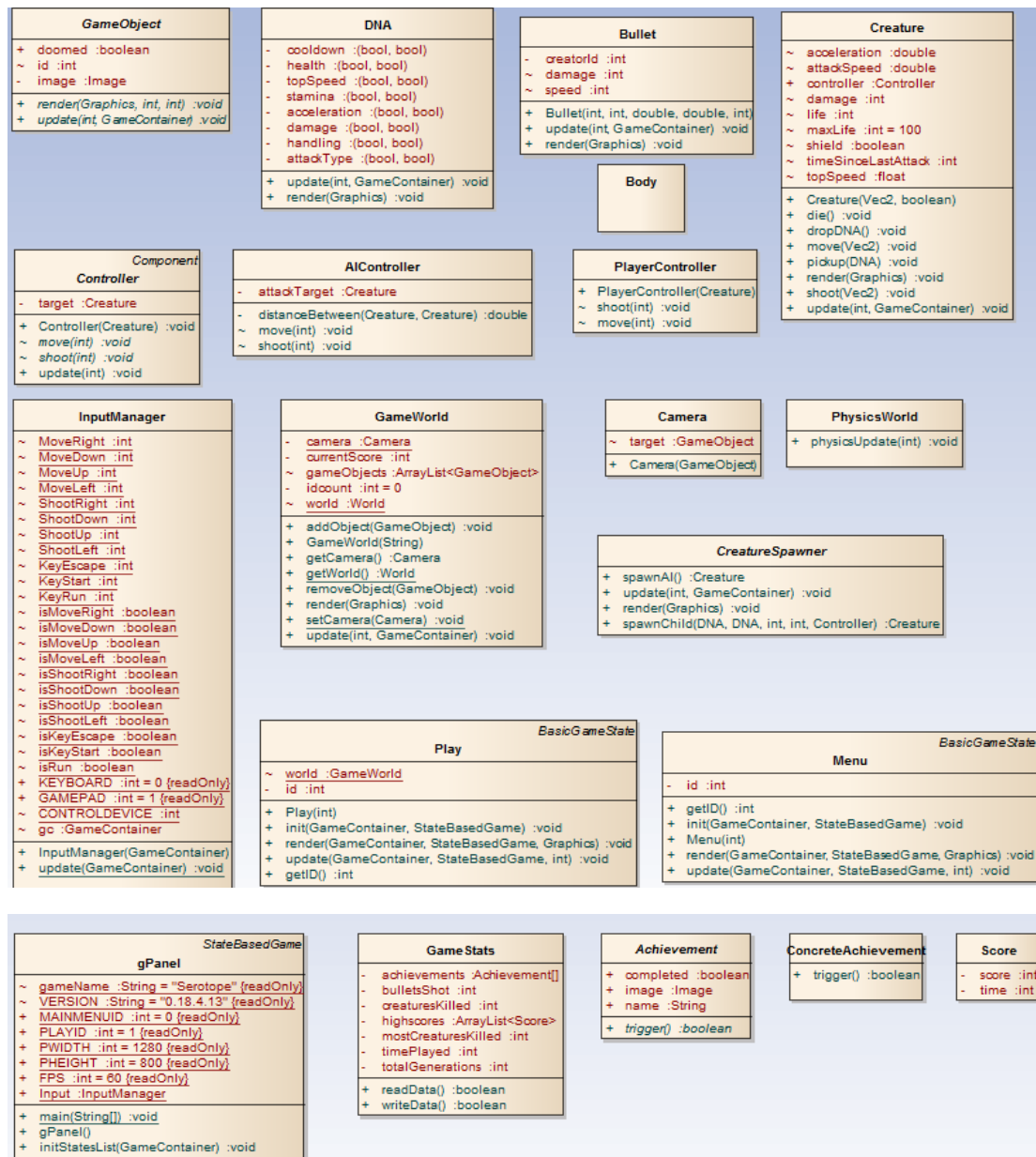
4.1.1 CLASS DIAGRAM OUTLINE

An outline of the class diagram with methods and attributes omitted is provided below:



4.1.2 CLASS DIAGRAM – CLASS ATTRIBUTES AND METHODS

Details on the attributes and methods for each class:



4.2. CLASS DICTIONARY

AI Controller – handles the behaviour of AI controlled creatures in the game.

Achievement – abstract class representing the different achievements that can be achieved throughout the game.

Body – represents the physical body of a game object in the game.

Bullet – keeps record of a projectile fired from a creature during the game.

Camera – the co-ordinates from which the game is rendered onto the screen.

Controller – abstract class providing the template for how creatures are controlled in the game.

Creature – The game units of the game controlled by the AI and player.

CreatureSpawner – Class that creates the game's creatures and enters them into the game.

DNA – Contains data of the creature's stats and "traits."

GameObject – Abstract class that encompasses all objects in the game such as "creatures", "bullets", "camera" and "DNA".

GameStats – Responsible for maintaining the persistent data of the game including achievements and highscores.

GameWorld – Represents the game environment in which the game is played.

InputManager – Responsible for collecting the input data from the user during the game.

Menu – handles the menu state including navigation.

PhysicsWorld – Handles the physics collisions between game objects in the game.

Play – Handles the Play game state.

PlayerController – the controller for the creature that is controlled by the user.

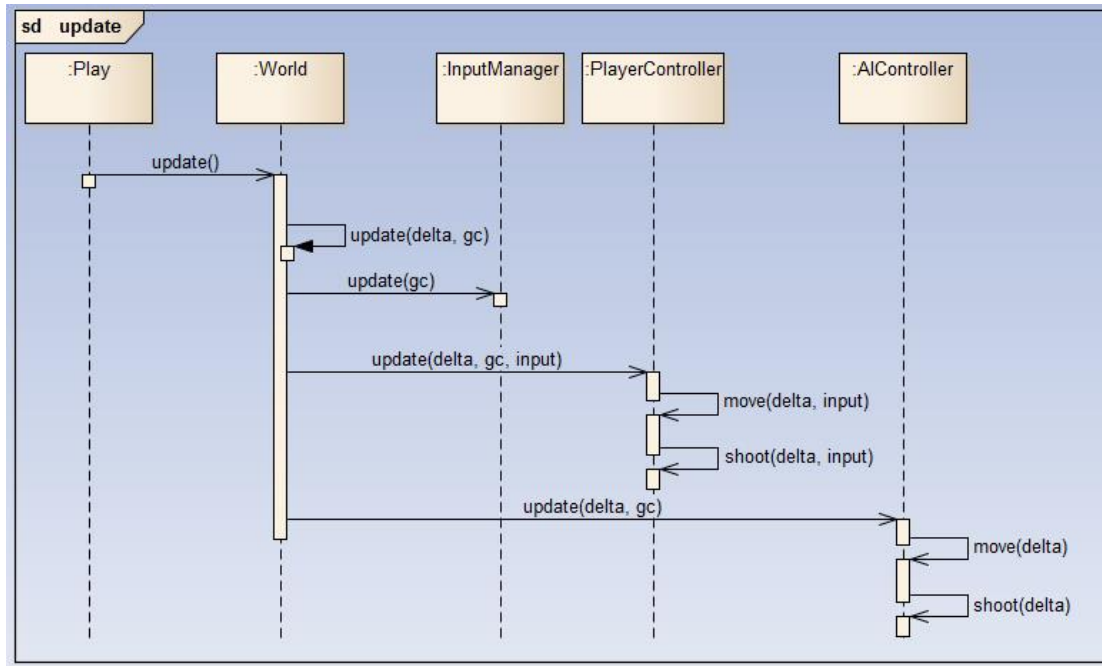
Score – contains data of a game score used by the GameStats class.

gPanel – The 'appGameContainer' managed by slick, it is the primary interface between Serotope and the slick library.

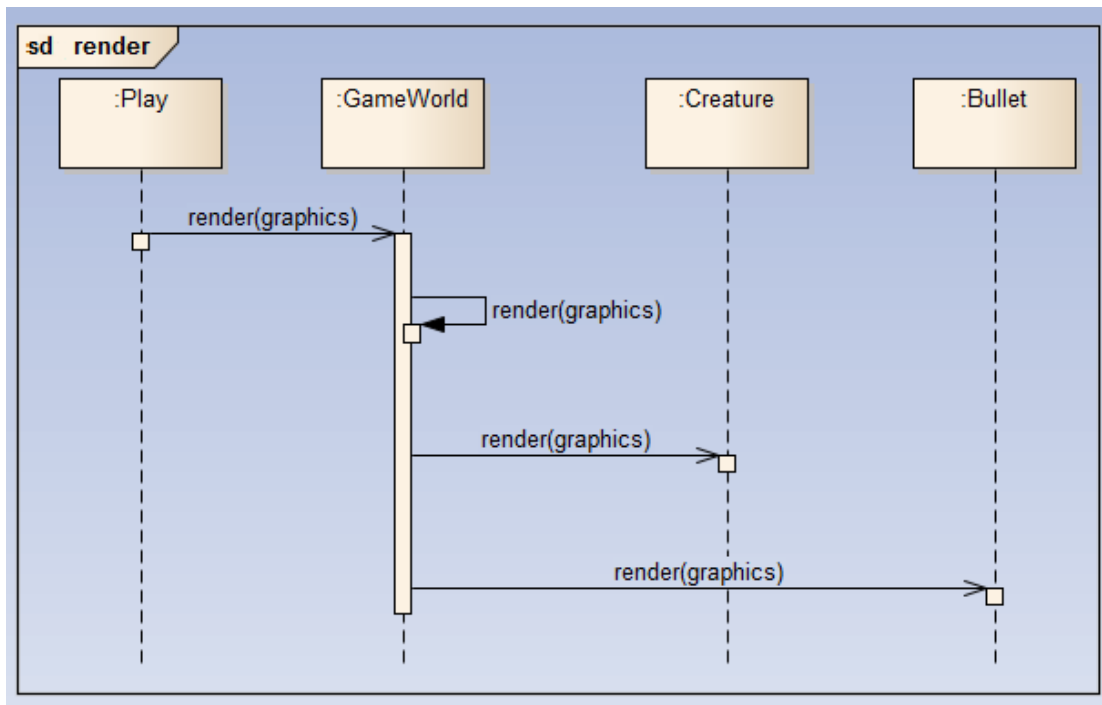
5. DYNAMIC MODEL

5.1 SEQUENCE DIAGRAMS

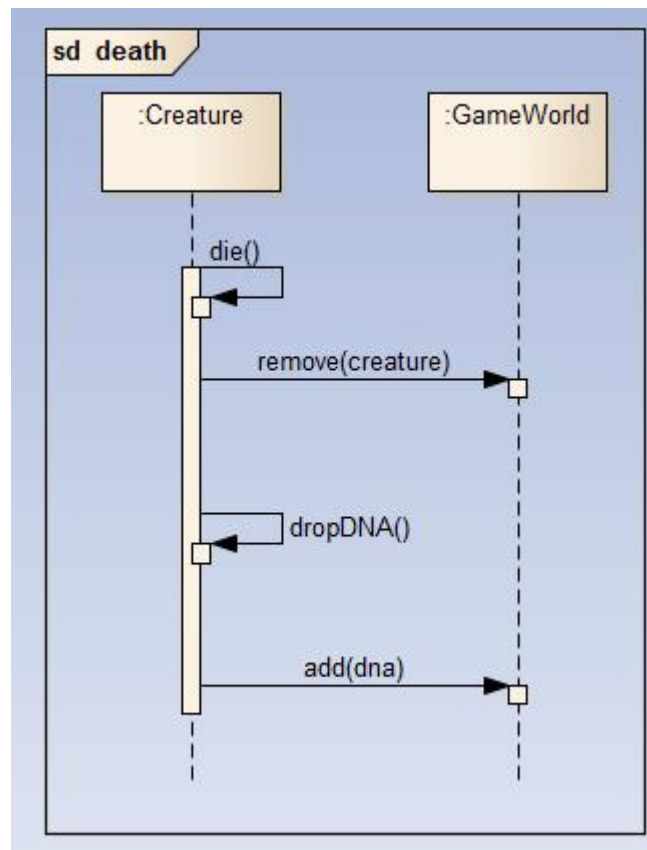
5.1.1 THE GAME UPDATE LOOP SEQUENCE DIAGRAM



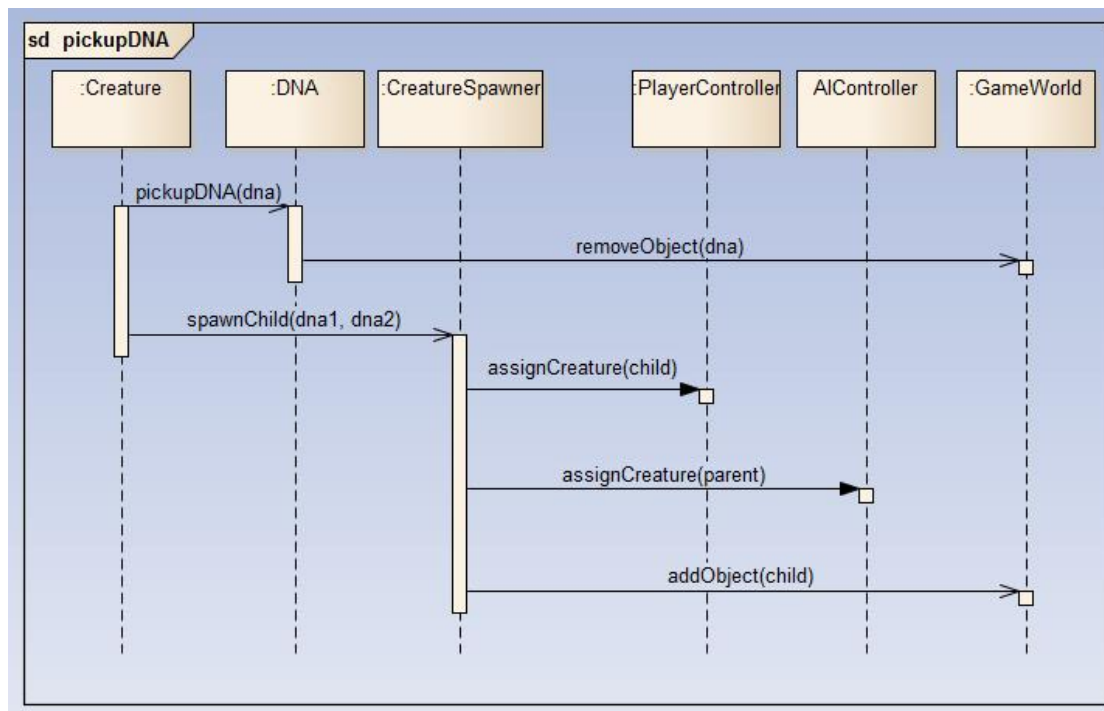
5.1.2 THE GAME RENDER LOOP SEQUENCE DIAGRAM



5.1.3 CREATURE DEATH SEQUENCE DIAGRAM



5.1.4 PICKUP DNA SEQUENCE DIAGRAM



5.2. ACTIVITY DIAGRAMS

5.2.1. MAIN MENU ACTIVITY DIAGRAM

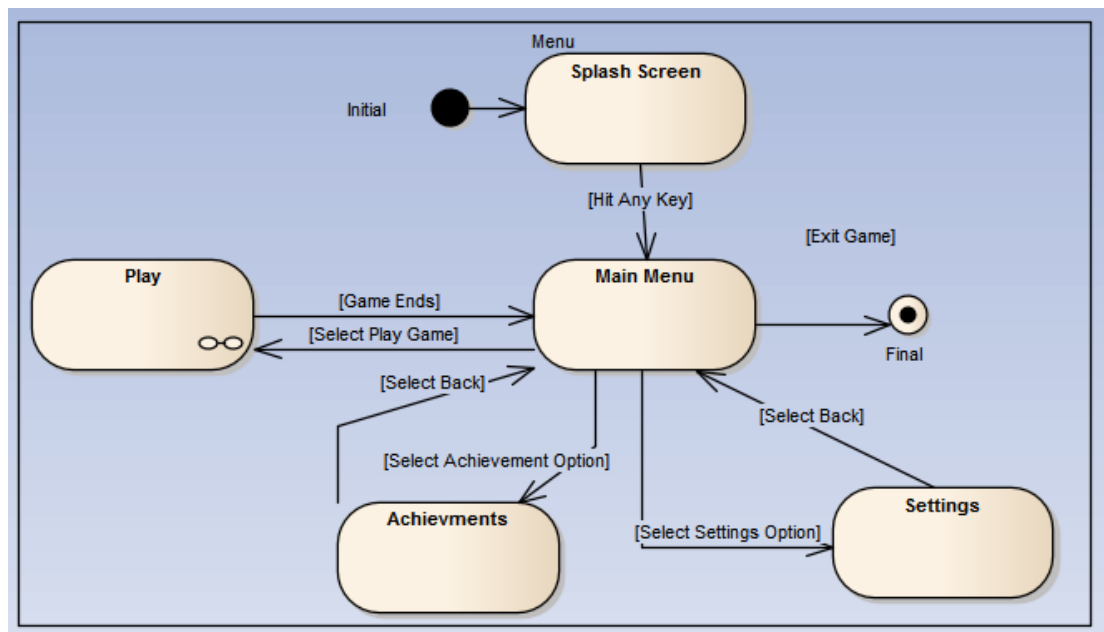
Due to size constraints of the document, this has been included as a .eap file in the submission, as well as in the appendix.

5.2.2. GAMEPLAY ACTIVITY DIAGRAM

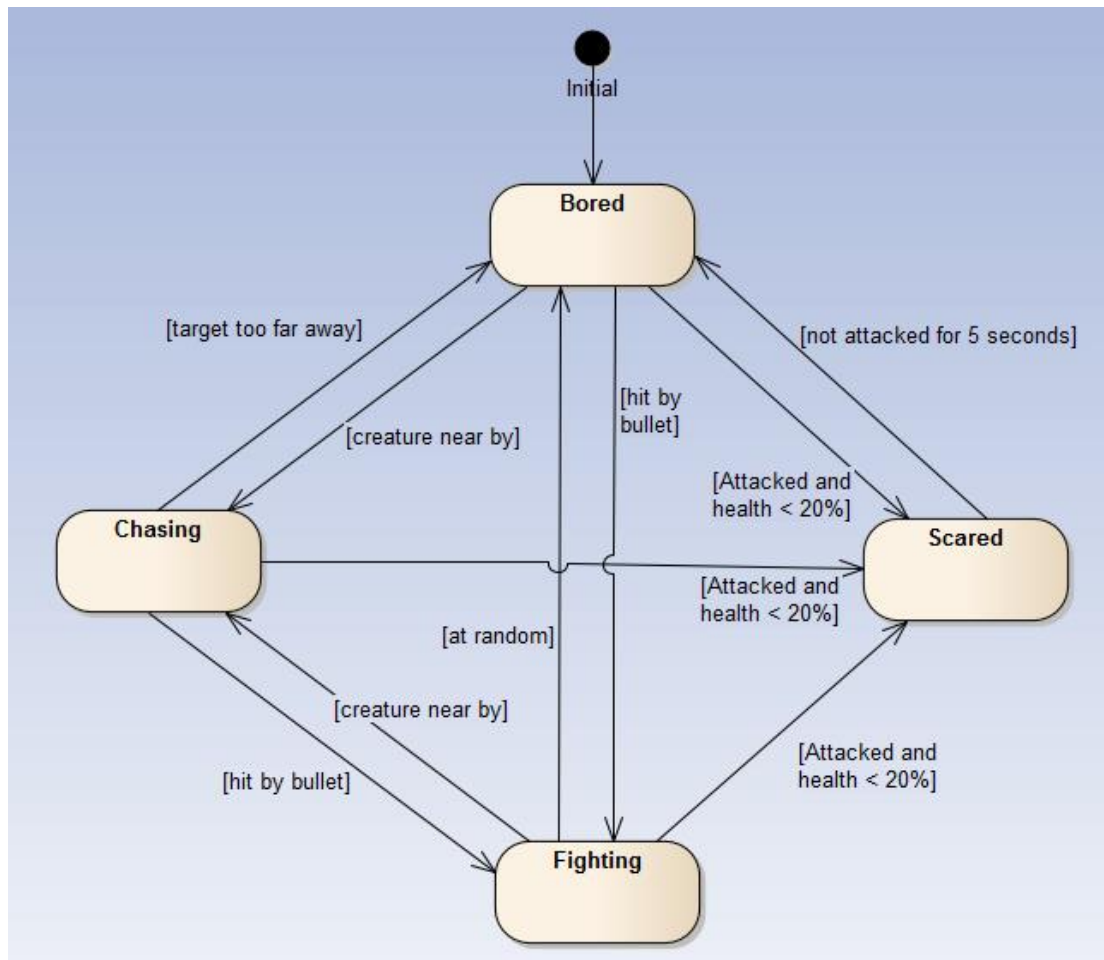
Due to size constraints of the document, this has been included as a .eap file in the submission, as well as in the appendix.

5.3. STATE DIAGRAMS

5.3.1. MENU STATE DIAGRAM



5.3.3 AI STATE DIAGRAM



6. DATA DESIGN

6.1. DATA DESCRIPTION

6.1.1. TEMPORARY DATA

6.1.1.1. BODY

Body objects are stored in both the physicsWorld object inside GameWorld and referenced inside their associated GameObject.

Body.getUserData() shall return the Body objects associated GameObject.

6.1.1.2. DNA

DNA objects are stored either in the GameObjects array inside gameWorld, or with a creature; never in both at the same time.

6.1.1.3. GAMEOBJECT

GameObject objects are stored in the GameObjects array in GameWorld.

The first object to be initialized shall be the user's creature, which will always be the first object in the gameObjects array.

6.1.2. PERMANENT DATA

6.1.2.1. GAMESTATS

GameStats is a singleton object that manages the permanent data stored by the system.

- The data to be stored is scores, achievements, and statistic counters for achievements.
- Both GameWorld and Menu have a reference to the GameStats object.
- At the end game state, GameStats.writeData() shall be called.
- On game startup, GameStats.readData() shall be called.

6.2. DATA DICTIONARY

6.2.1. BODY

A Body is a JBOX2d object that populates the physicsWorld, and contains the physical attributes of an object.

The JBOX2d javadoc can be found at:

<https://code.google.com/p/jbox2d/downloads/detail?name=jbox2d-library-2.1.2.0-javadoc.jar>

6.2.2. DNA

The DNA class consists of an attribute for each of the traits in the game. These are in the form of a tuple of two Booleans. The tuple represents a pair of alleles, one from each of its parent's DNA. The two possible alleles in a gene are "has the trait" and "doesn't have the trait", in which "has the trait" is always recessive and represented by true (whereas "doesn't have the trait", is always dominant and represented by false).

In addition to inherited Data and Methods from GameObject, DNA has 9 attributes, and no methods.

These attributes are:

```
(Boolean, Boolean) life
(Boolean, Boolean) stamina
(Boolean, Boolean) shield
(Boolean, Boolean) attackSpeed
(Boolean, Boolean) damage
(Boolean, Boolean) shootingType
(Boolean, Boolean) speed
(Boolean, Boolean) acceleration
(Boolean, Boolean) handling
```

For reasoning regarding to the design of DNA and its functional requirements refer to section 8.1.

6.2.3. GAMEOBJECT

GameObject is the fundamental component of the Game World.

It has 4 components:

Image `image`

This is used by the GameObject's `render(graphics, x, y)` method to draw the image on screen at position `x y`.

int `id`

This is the GameObjects unique identifier which is assigned via a call to the `GameWorld` singleton.

Body `body`

This is the object's 'Physical' representation used in physics calculations. It also contains information on its position.

boolean `solid`

This is a flag that determines whether the object takes part in physics calculations or is 'ethereal', like a `Marker` or `CreatureSpawner`.

GameObject defines 2 methods:

abstract public void `update(int delta, GameContainer gc)`

This is called on every time step to update the game object according to some delta.

void `render(Graphics g, float xrender, float yrender)`

This is called on every time the object is to be drawn, it shall be the same for all game objects.

6.2.4. GAMESTATS

GameStates is an object used to store permanent data relating to the game's statistics.

It has 4 components:

Achievement[] `achievements`

This is where all achievement data is stored.

ArrayList<Scores> `highScores`

This is where all highScore information is stored.

int `bulletsShot`

Is the total number of bullets shot by the users of this game in total.

int `timePlayed`

Is the total time played by the users of this game.

int `creaturesKilled`

Is the number of creatures killed by users of this game in total.

GameStats defines 2 methods:

```
boolean public readData()
```

This is to be called every time the game wants to initialize gameStats.

```
boolean public writeData()
```

This is to be called before the game exits to save gameStats.

7. HUMAN INTERFACE DESIGN

7.1 USER INTERFACE SUMMARY

The game will consist of two main types of user interface: menu and gameplay. In the menu sections, the user will navigate using the mouse to click on labeled options. In the gameplay section the player will use 10 keys to play the game. The game controls can be customised in the Settings menu. The default layout will be W, A, S, D to move up, left, down and right respectively. UP ARROW, LEFT ARROW, DOWN ARROW and RIGHT ARROW will be the default keys for shooting up, left, down and right respectively. SHIFT is used to make the player creature sprint and ESC pauses the game.

7.2 SCREEN IMAGES

7.2.1 SPLASH SCREEN PROTOTYPE

Serotope

<Press any key>

7.2.2 MAIN MENU PROTOTYPE

Serotope

Game Title

Play
How to
Achievements

Menu Options

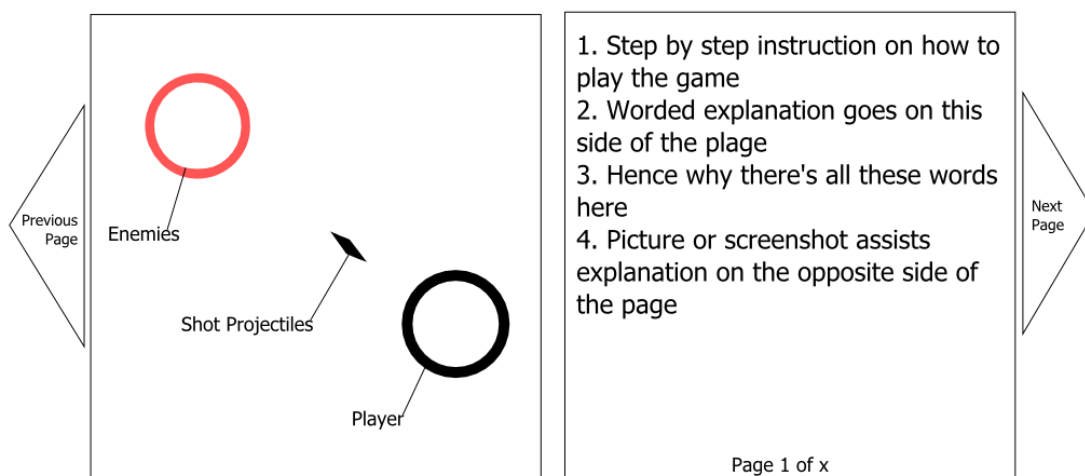
Settings



7.2.3 TUTORIAL PROTOTYPE

How To Play

Menu title



















Go Back

Achievements

Highscores

1	Andy	1070
2	Jo	1003
3	Mike	999
4	Trish	977
5	Ben	950
6	Andy	943
7	Nicky	942
8	Suzy	925
9	Bob	920
10	Anna	889

Medals

			
			
			
			
Medal explanation			

[Go Back](#)


Settings

Menu title

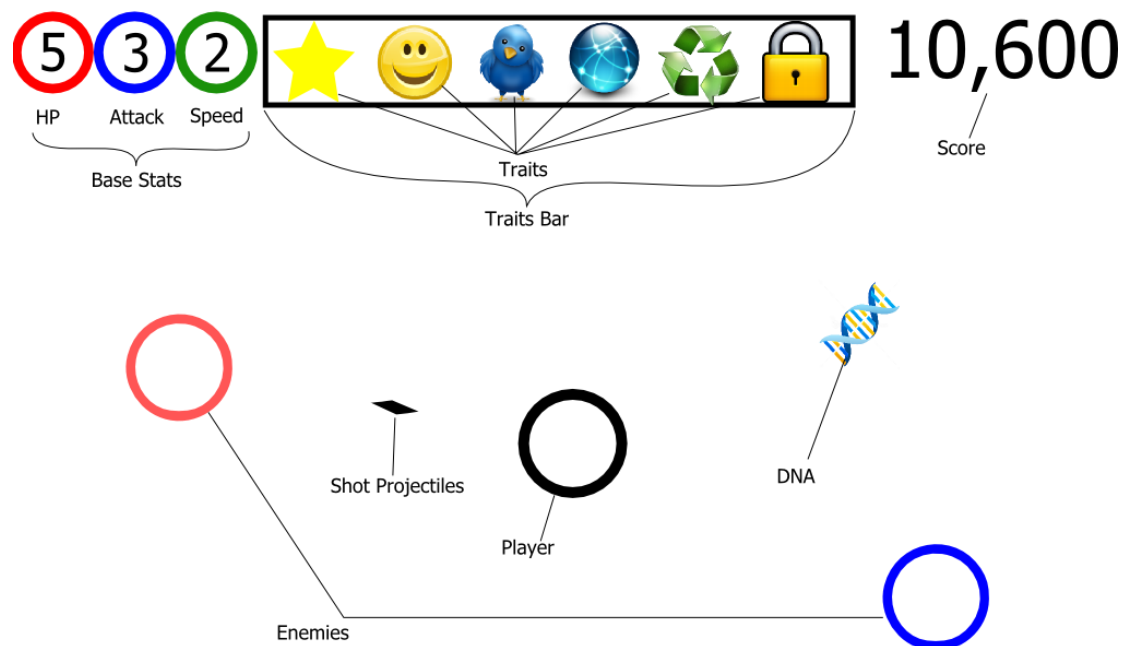
Volume



Controls

	Move Left		Shoot Left		Pause Game
	Move Right		Shoot Right		Sprint
	Move Up		Shoot Up		
	Move Down		Shoot Down		Go Back

7.2.6 GAMEPLAY PROTOTYPE



7.2.7 PAUSE MENU PROTOTYPE

Paused

Score 10,800

Resume

Return to Main Menu

Game Over

Final Score 10,800

Replay

Return to Main Menu

7.3. SCREEN OBJECTIVES AND ACTIONS

7.3.1. SPLASH SCREEN

The splash screen will be loaded when the user opens the program. The user enters the Main Menu section from here by simply pressing any key, which the instruction prompts them to do.

7.3.2. MAIN MENU

The main menu provides the user access to all the features of the program. There are 4 four options: Play Game, Tutorial, Achievements or Settings. Every option is labeled except for the Settings, which is the cog symbol at the bottom right of the screen. Each option is accessed by mouse clicking on the button.

7.3.3. TUTORIAL MENU

The tutorial menu provides an explanation of the game to the user. Text instructions are displayed on the right half of the screen with an assisting picture on the left hand side. The user navigates the tutorial by mouse clicking on the triangles on the screen edges labeled “Next Page” and “Previous Page”. To return to the Main Menu the user must mouse click of the “Go Back” option at the bottom right hand of the screen.

7.3.4. ACHIEVEMENTS MENU

The achievements menu displays the top 10 scores of the game in ranked order from highest to lowest on the left half of the screen. On the right hand side the medal table is displayed. The user can mouse click on each medal box and the text box below will give an explanation on how the medal can be unlocked. If the medal has been achieved, the medal image will be displayed; otherwise a locked symbol will be shown in the box instead. To return to the main menu the user can mouse click the “Go Back” option.

7.3.5. SETTINGS MENU

In the settings menu, the user can customise some of the game settings. This consists of editing the game volume displayed as a sound bar. The user can mouse click or drag anywhere on the bar to set the volume. The user is also able to edit the gameplay controls here. To change a key setting, they need to mouse click on the box where the current key being used is, and then press the key they would prefer to use instead. To return to the main menu the user can mouse click the “Go Back” option.

7.3.6. GAMEPLAY

In the game the user controls a single creature, which they can move using the W, A, S and D keys, shoot projectiles with UP ARROW, LEFT ARROW, DOWN ARROW and RIGHT ARROW and sprint with SHIFT, unless they have customized the controls in the settings menu. The user can also pause the game at any point using the default ESC key. The player creature picks up DNA automatically by moving within a set radius of that object. Creatures cannot move through other creatures and the world contains no walls, instead the world edges wrap around to each other. The camera for the game is centered on the player controlled creature so it is typically rendered at the middle of the screen.

7.3.7. PAUSE MENU

The pause menu displays the player’s current score and allows two options that they can select by mouse clicking on them. They can either choose the “Resume” option where they leave the pause menu and return to the current game, or they can choose to “Return to Main Menu” which returns the player to the main menu screen and the current game’s progress is ended.

7.3.8. GAME OVER MENU

This menu displays the final score of that game and 2 options. The user can either choose to “Play Again” which starts a new game or to “Return to Main Menu” which will take them back to the main menu

8. APPENDICES

8.1. DESIGN OF DNA WITHIN THE SEROTOPE GAME

The game aims to teach the User about Mendelian inheritance, in which the two main principles are:

1. Every individual possesses a pair of alleles for any particular trait and that each parent passes a randomly selected copy of only one of these alleles to its offspring.
2. The alleles that are inherited from each parent is independent of one another (i.e. the selection of one trait will not impact on the selection of another).

To show these principles in the game, whenever a creature dies, it leaves behind its DNA. This can then be picked up by the User, causing it to be combined with its own DNA, and spawning a child based on the results.

The DNA object in the game is supposed to be a basic representation of real world DNA, consisting of various genes, which each corresponds to a trait. These genes are represented as a pair of alleles, which can be either dominant or recessive.

The traits that are to be included in the game are the following:

- Life
Health points
- Stamina
The rate at which life runs out
- Shield
How many bullets can be absorbed before Life begins to be affected
- Attack Speed
The speed at which a creature can shoot bullets
- Damage
The amount of damage a bullet will do on impact
- Shooting Type
The way a creature 'shoots'
- Speed
The maximum speed a creature can move
- Acceleration
The time it takes for a creature to reach maximum speed
- Handling
How long it takes for a creature to stop moving

Each trait is represented in DNA as a 'gene', consisting of a pair of alleles. There are two possible alleles for each trait, with one being dominant, and the other recessive. It is the combination of two alleles in a gene that determine the final trait.

For example, the health gene consists of a recessive allele 'healthy', and the dominant allele 'weak'. These can be represented as h and W respectively.

If a creature has a pair of alleles that are WW, Wh, or hW, they get the 'weak' trait.
If the creature has the pair of alleles hh, they get the 'healthy' trait, and have extra health.

However, this poses a problem, in which a creature can either have high or low health, but no in between. To overcome this issue, there can be multiple genes for the same trait, which add their bonuses together.

class System

```

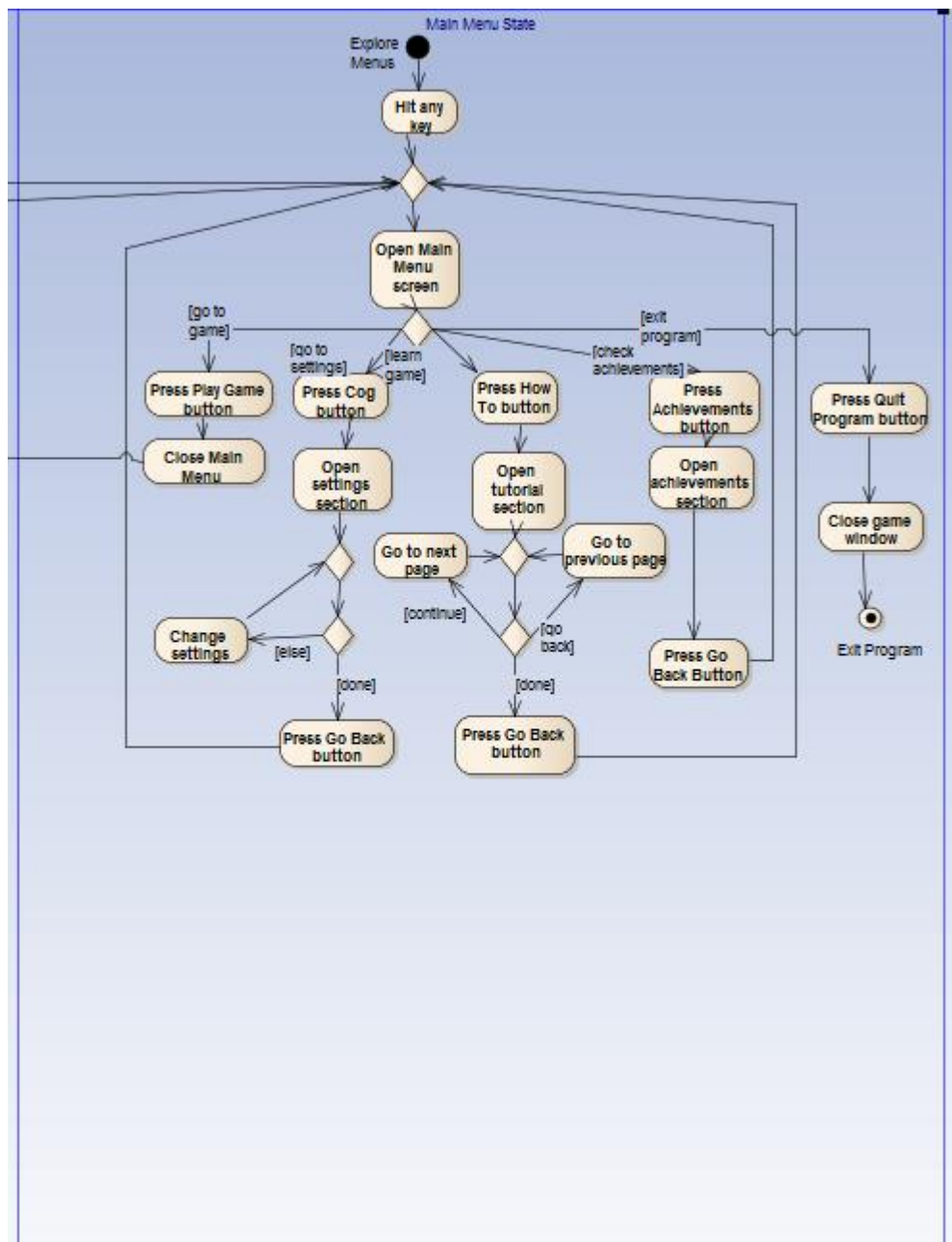
classDiagram
    class DNA {
        -cooldown : (bool, bool)
        -health : (bool, bool)
        -topSpeed : (bool, bool)
        -stamina : (bool, bool)
        -acceleration : (bool, bool)
        -damage : (bool, bool)
        -handling : (bool, bool)
        -attackType : (bool, bool)
        +update(int, GameContainer) : void
        +render(Graphics) : void
    }
    class GameObject {
        +doomed : boolean
        -id : int
        -image : Image
        +render(Graphics, int, int) : void
        +update(int, GameContainer) : void
    }
    class Body {
    }
    class AIController {
        -attackTarget : Creature
        +distanceBetween(Creature, Creature) : double
        +move(int) : void
        +shoot(int) : void
    }
    class Creature {
        -acceleration : double
        -attackSpeed : double
        -controller : Controller
        -damage : int
        -life : int
        -maxLife : int = 100
        -shield : boolean
        -timeSinceLastAttack : int
        -topSpeed : float
        +Creature(Vec2, boolean)
        +die() : void
        +drop(DNA) : void
        +move(Vec2) : void
        +pickup(DNA) : void
        +render(Graphics) : void
        +shoot(Vec2) : void
        +update(int, GameContainer) : void
    }
    class Bullet {
        -creatorId : int
        -damage : int
        -speed : int
        +Bullet(int, int, double, double, int)
        +update(int, GameContainer) : void
        +render(Graphics) : void
    }
    class Camera {
        ~target : GameObject
        +Camera(GameObject)
    }
    class CreatureSpawner {
        +spawnAI() : Creature
        +update(int, GameContainer) : void
        +render(Graphics) : void
        +spawnChild(DNA, DNA, int, int, Controller) : Creature
    }
    class GameWorld {
        -camera : Camera
        -currentScore : int
        -gameObjects : ArrayList<GameObject>
        -icountdown : int = 0
        ~world : World
        +addObject(GameObject) : void
        +GameWorld(String)
        +getCamera() : Camera
        +getWorld() : World
        +removeObject(GameObject) : void
        +render(Graphics) : void
        +setCamera(Camera) : void
        +update(int, GameContainer) : void
    }
    class PhysicsWorld {
        +physicsUpdate(int) : void
    }
    class InputManager {
        ~MoveRight : int
        ~MoveDown : int
        ~MoveUp : int
        ~MoveLeft : int
        ~ShootRight : int
        ~ShootDown : int
        ~ShootUp : int
        ~ShootLeft : int
        ~KeyEscape : int
        ~KeyStart : int
        ~KeyRun : int
        ~isMoveRight : boolean
        ~isMoveDown : boolean
        ~isMoveUp : boolean
        ~isMoveLeft : boolean
        ~isShootRight : boolean
        ~isShootDown : boolean
        ~isShootUp : boolean
        ~isShootLeft : boolean
        ~isKeyEscape : boolean
        ~isKeyStart : boolean
        ~isRun : boolean
        +KEYBOARD : int = 0 (readOnly)
        +GAMEPAD : int = 1 (readOnly)
        +ANDROID : int = 2 (readOnly)
        ~CONTROLDEVICE : int
        ~gc : GameContainer
        +InputManager(GameContainer)
        +update(GameContainer) : void
    }
    class GameStats {
        -achievements : Achievement[]
        -bulletsShot : int
        -creaturesKilled : int
        -highscores : ArrayList<Score>
        -mostCreaturesKilled : int
        -timePlayed : int
        -totalGenerations : int
        +readData() : boolean
        +writeData() : boolean
    }
    class Achievement {
        +completed : boolean
        +image : Image
        +name : String
        +trigger() : boolean
    }
    class ConcreteAchievement {
        +trigger() : boolean
    }
    class Score {
        -score : int
        -time : int
    }
    class Menu {
        -id : int
        +getID() : int
        +init(GameContainer, StateBasedGame) : void
        +Menu(int)
        +render(GameContainer, StateBasedGame, Graphics) : void
        +update(GameContainer, StateBasedGame, int) : void
    }
    class gPanel {
        -gameName : String = "Srotopote" (readOnly)
        ~VERSION : String = "0.18.4.13" (readOnly)
        ~MAINMENUID : int = 0 (readOnly)
        ~PLAYID : int = 1 (readOnly)
        ~PWIDTH : int = 1280 (readOnly)
        ~PHEIGHT : int = 800 (readOnly)
        ~FPS : int = 60 (readOnly)
        ~Input : InputManager
        +main(String[]) : void
        +gPanel()
        +initStatesList(GameContainer) : void
    }
    class StateBasedGame {
    }
    class BasicGameState {
    }

    DNA --> GameObject
    GameObject --> Body
    AIController --> Creature
    Creature --> Bullet
    Camera --> GameObject
    CreatureSpawner --> GameWorld
    GameWorld --> PhysicsWorld
    GameWorld --> GameStats
    GameWorld --> Menu
    GameStats --> Achievement
    GameStats --> ConcreteAchievement
    GameStats --> Score
    InputManager --> GameWorld
    InputManager --> gPanel
    gPanel --> StateBasedGame
    StateBasedGame --> BasicGameState
    BasicGameState --> Menu
    Menu --> BasicGameState
    
```

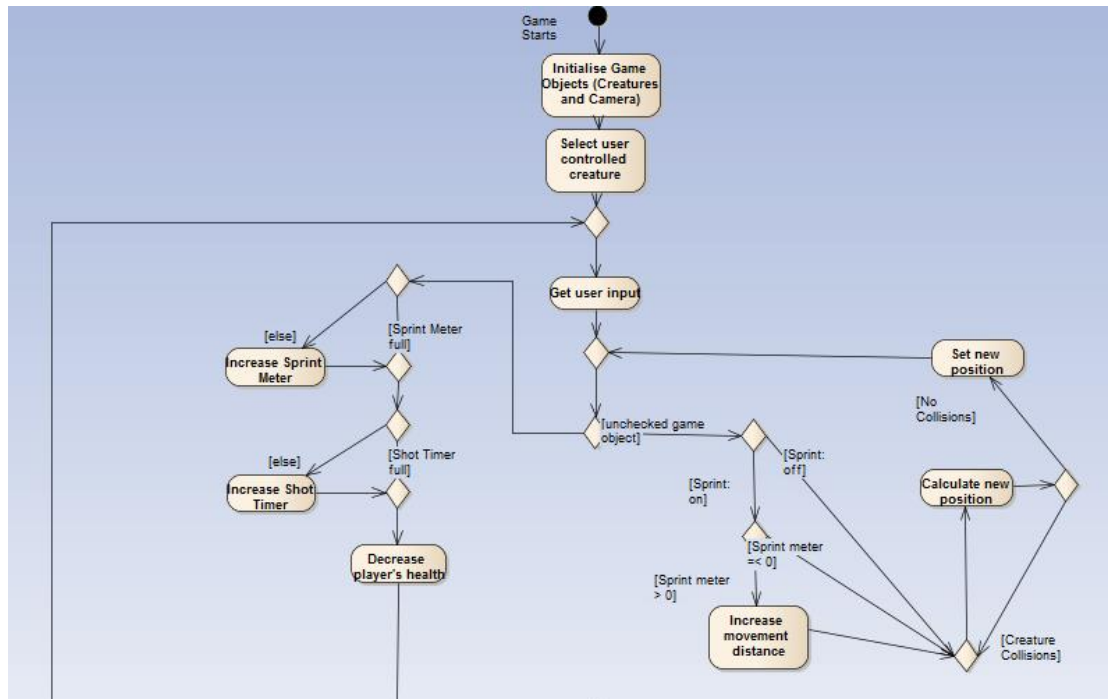
The diagram illustrates the architecture of a game system, organized into several functional layers and components:

- Core Entities & Physics:**
 - DNA:** Defines creature attributes (cooldown, health, topSpeed, stamina, acceleration, damage, handling, attackType) and methods for updating and rendering.
 - GameObject:** A base entity with a doomed status, ID, image, and methods for rendering and updating.
 - Body:** A simple entity class.
 - Bullet:** Represents projectiles with creator ID, damage, speed, and methods for creation, updating, and rendering.
 - Camera:** Manages the view, holding a target GameObject and a reference to the GameWorld.
 - PhysicsWorld:** Handles physics calculations, updating the GameWorld.
- Game Management & State:**
 - GameWorld:** The central hub managing the game state. It holds the camera, current score, a list of game objects, an icountdown timer, and a reference to the world. It provides methods for adding/removing objects, rendering, and updating.
 - GameStats:** Tracks game progress, including achievements, bullets shot, creatures killed, highscores, most creatures killed, time played, and total generations. It includes methods for reading and writing data.
 - Achievement:** Defines an achievement with a completion status, image, name, and a trigger method.
 - ConcreteAchievement:** Implements the trigger method for a specific achievement.
 - Score:** Represents a score with its value and the time taken to achieve it.
 - Menu:** Manages the game's menu system, including ID, initialization, rendering, and updating.
- Player & Input Handling:**
 - AIController:** Manages AI behavior, including attack targets, distance calculations, movement, and shooting.
 - Controller:** A base controller class with target, movement, shooting, and updating methods.
 - PlayerController:** Implements the Controller interface for player actions.
 - InputManager:** Processes user input from various sources (KEYBOARD, GAMEPAD, ANDROID) and maps it to game actions like movement and shooting.
- Game Presentation & Initialization:**
 - gPanel:** The main game panel, initialized with game name, version, menu IDs, dimensions, FPS, and an input manager. It provides methods for main execution and state initialization.
 - StateBasedGame:** An interface or base class for game states.
 - BasicGameState:** A concrete implementation of the StateBasedGame interface.

8.3.2 ACTIVITY DIAGRAM – MENU NAVIGATION (PART 2/2)



8.4.1 ACTIVITY DIAGRAM – GAMEPLAY (PART 1/2)



8.4.2 ACTIVITY DIAGRAM – GAMEPLAY (PART 2/2)

