Serotope Software Design Document

Benedict Hobart Leanne Capewell Jordan Burdett

Bhobart LEanneec JBurdett

|  |
| --- |
| Table of Contents  1. **Introduction**    1. Purpose    2. Scope    3. Overview    4. Definitions and Acronyms 2. **Use Cases**    1. Actors    2. List of Use Cases    3. Use Case Diagrams    4. Use Cases 3. **Analysis Overview**    1. Component Diagram    2. Domain Model    3. Subsystems    4. Subsystem Interfaces 4. **Design Overview**    1. Classes    2. Class Dictionary 5. **Dynamic Model**    1. Sequence Diagrams    2. Activity Diagrams    3. State Diagrams 6. **Data Design**    1. Data Description    2. Data Dictionary 7. **Human Interface Design**    1. Overview of Human Interface    2. Screen Images    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 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***
* **Add more as we add more to the design document**

# 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 that all users will perform similar actions.

### 2.1.2 Database

The database is the local data saved on the Users 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

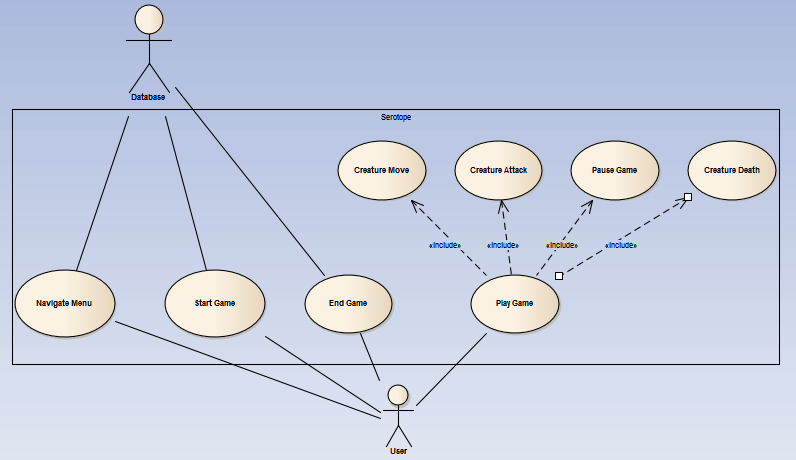
2.2.6.Creature Death

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  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  Database  **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  Database  **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  Database  **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  Database  **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) |

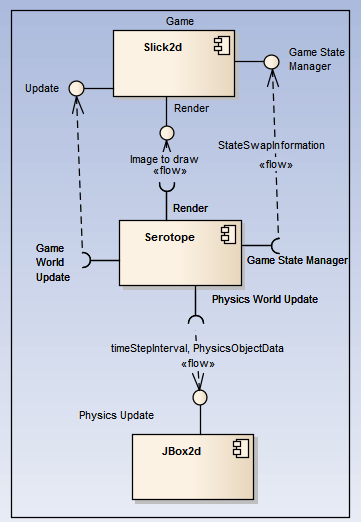
|  |
| --- |
| 2.4.7. Pick up DNA **Actors**  User  Database  **Stakeholders and their needs**  User – Wants to evolve their creature into one with better traits.  **Preconditions**  User has started a game and it is currently being displayed on the screen  **Postconditions**  None  **Trigger**  User's 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 |

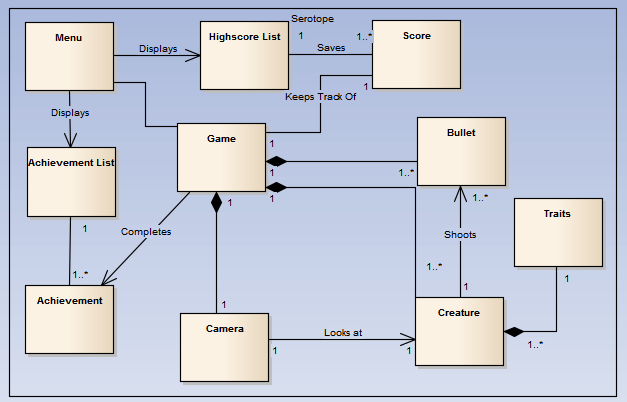
|  |
| --- |
| 2.4.9. End game **Actors**  User  Database  **Stakeholders and their needs**  User – wants to stop playing the game  **Preconditions**  User has started a game and it is currently being displayed on the screen  **Postconditions**  High score screen is displayed.  **Trigger**  User's creature's 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, along with the high score list   **Extensions**  None |

# 3. AnalysisOverview

## 3.1. Component Diagram



## Domain Model



## 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 a Low-level frameworks for playing audio, drawing graphics and detecting user input.

### 3.3.3. JBox2d

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

## Subsytem Interfaces

Slick provides Serotope with

* A game state manager, to swap between the Game and Menu States.
* A update function, that creates a way for Serotope to update at approximately regular intervals, as well as the time between those update intervals, used for ins game calculation.
* A render function, that creates a graphics object that will be drawn to the screen in at, 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

# 4. Design Overview

## 4.1. Class diagram

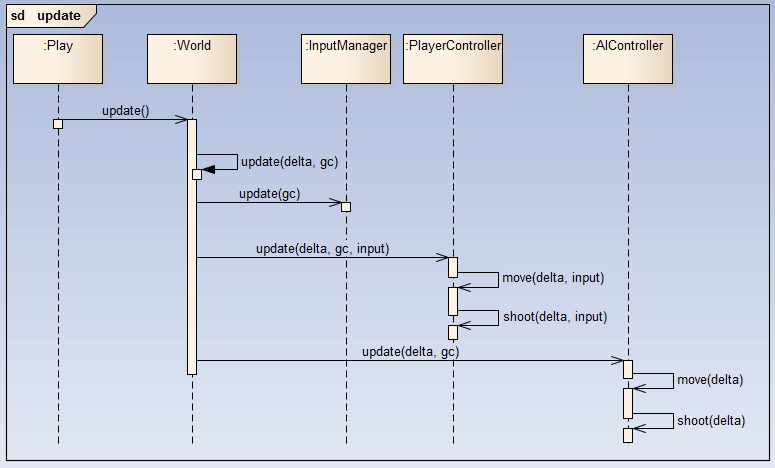
## 4.2. Class dictionary

# 5. DynamicModel

## 5.1 Sequence Diagrams

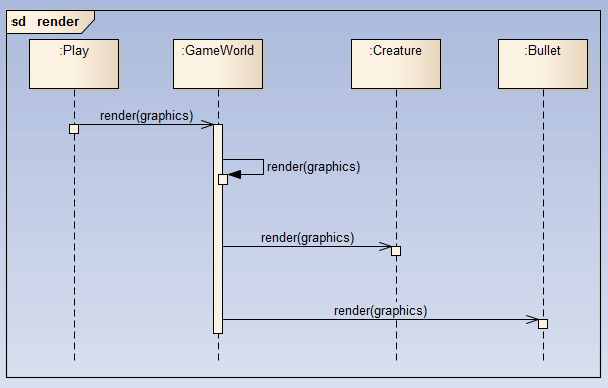
### 5.1.1 The game update loop Sequence Diagram

This diagram shows an overview of the game update loop, which repeatedly causes the game world and game objects to update based on changes in the game state.



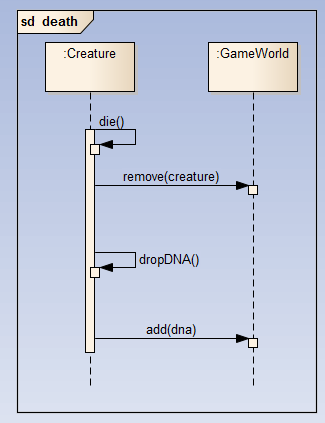
### 5.1.2 The game render loop Sequence Diagram

This diagram shows an overview of the game update loop, which repeatedly causes the game objects to render.



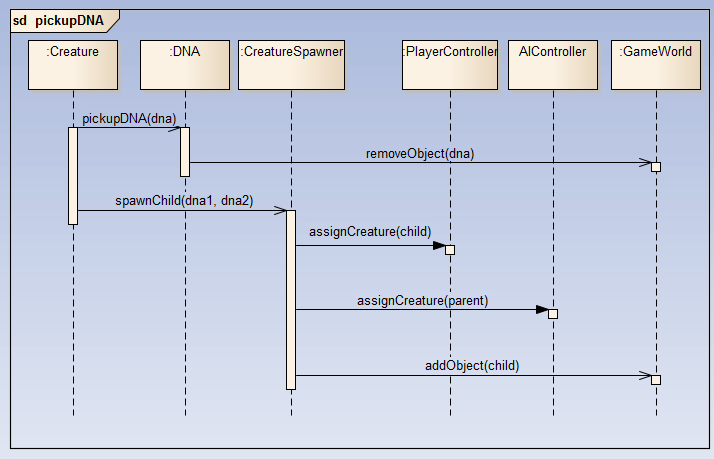
### 5.1.3 ­­Creature death Sequence Diagram

This diagram shows the interactions between a creature and the game world that occur when a creature dies in the game.



### 5.1.4 pickup dna sequence diagram

This diagram shows the process of a creature picking up a piece of DNA, causing a new creature to spawn, and be placed in control of the player.



## 5.2. ACTIVITY Diagrams

### 5.2.1. Main menu activity diagram

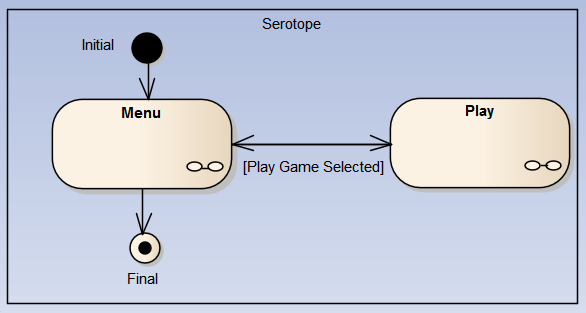
This diagram shows the various possible flows that can occur when a user interacts with the main menu options.

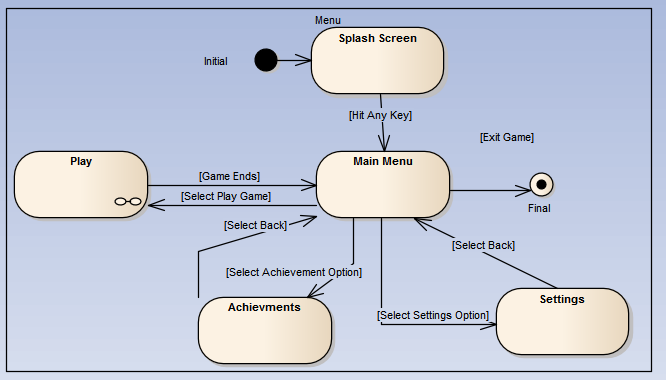
### 5.2.2. gameplay activity diagram

This diagram shows the various possible flows that can occur while a user is playing the game.

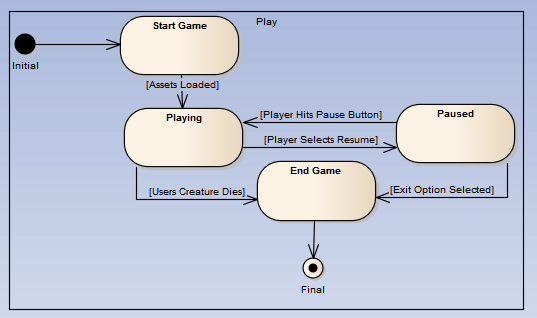
## 5.3. State Diagrams

### 5.3.1. Menu state diagram

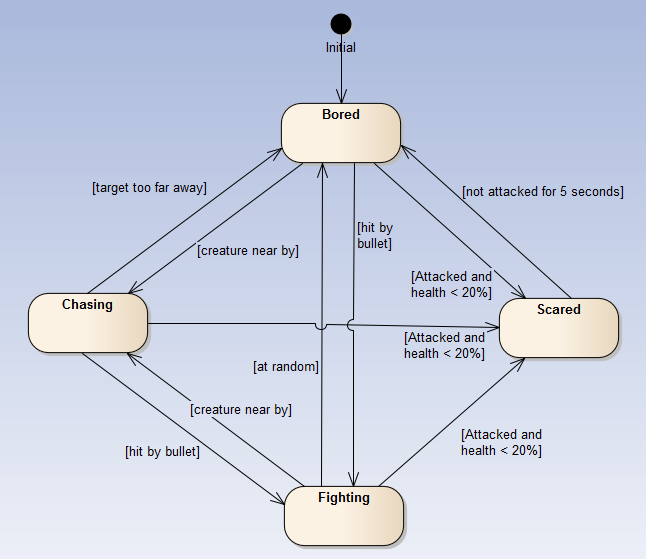




### 5.3.2. GamePlay state diagram



### 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, this populates the physicsWorld. Specifics of implantation are left to the developer.

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).

Ignoring inherited Data and Methods from GameObject

DNA has 9 Components

(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;

DNA has no methods

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

#### 6.2.3. Gameobject

The fundamental component to the Game World

GameObject 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 it is assigned via a call to the GameWorld singleton.

Body body;

This is the objects ‘Physical’ representation used in physics calculations, 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 monster spawner.

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

# 7. Human Interface Design

## 7.1. Overview of Human Interface

## 7.2. Screen Images

### 7.2.1.Main menu prototype

## 7.3. Screen Objectives and Actions

### 7.3.1.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:

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.

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, which 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
  + More or less 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 to 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.