|  |
| --- |
| Technical Design Document |
| Dinosaur Simulator |
| Richard Delamore |

|  |
| --- |
|  |

Contents

[1.0 Revision History Version 2](#_Toc15394261)

[2.0 Development Environment 2](#_Toc15394262)

[2.1 Game Engine 2](#_Toc15394263)

[2.2 IDE 2](#_Toc15394264)

[2.3 Source Control procedures 2](#_Toc15394265)

[2.4 Third Party Libraries 2](#_Toc15394266)

[2.5 Other Software 2](#_Toc15394267)

[3.0 Game Overview 3](#_Toc15394268)

[4.0 Graphics 4](#_Toc15394269)

[5.0 Artificial Intelligence 5](#_Toc15394270)

[6.0 Physics 8](#_Toc15394271)

[7.0 Items Item 8](#_Toc15394272)

[8.0 Game Flow 8](#_Toc15394273)

[9.0 Levels 8](#_Toc15394274)

[10.0 Interface 9](#_Toc15394275)

[10.1 Menu 9](#_Toc15394276)

[10.2 Camera 9](#_Toc15394277)

[10.3 Controls 9](#_Toc15394278)

[11.0 Asset List 9](#_Toc15394279)

[12.0 Technical Risks 9](#_Toc15394280)

# Revision History Version

|  |  |
| --- | --- |
| Version | Description |
| 1.0 | Initial Document |
| 1.1 | Update the section on the Dijkstra’s Algorithm |
| 2.0 | Massive changes the the Class diagram and Finite State Machines. I also removed the feature Footsteps and Sniffing due to lack of time. |

# Development Environment

## Game Engine

* Bootstrap

## IDE

* Visual Studio.

## Source Control procedures

* Github

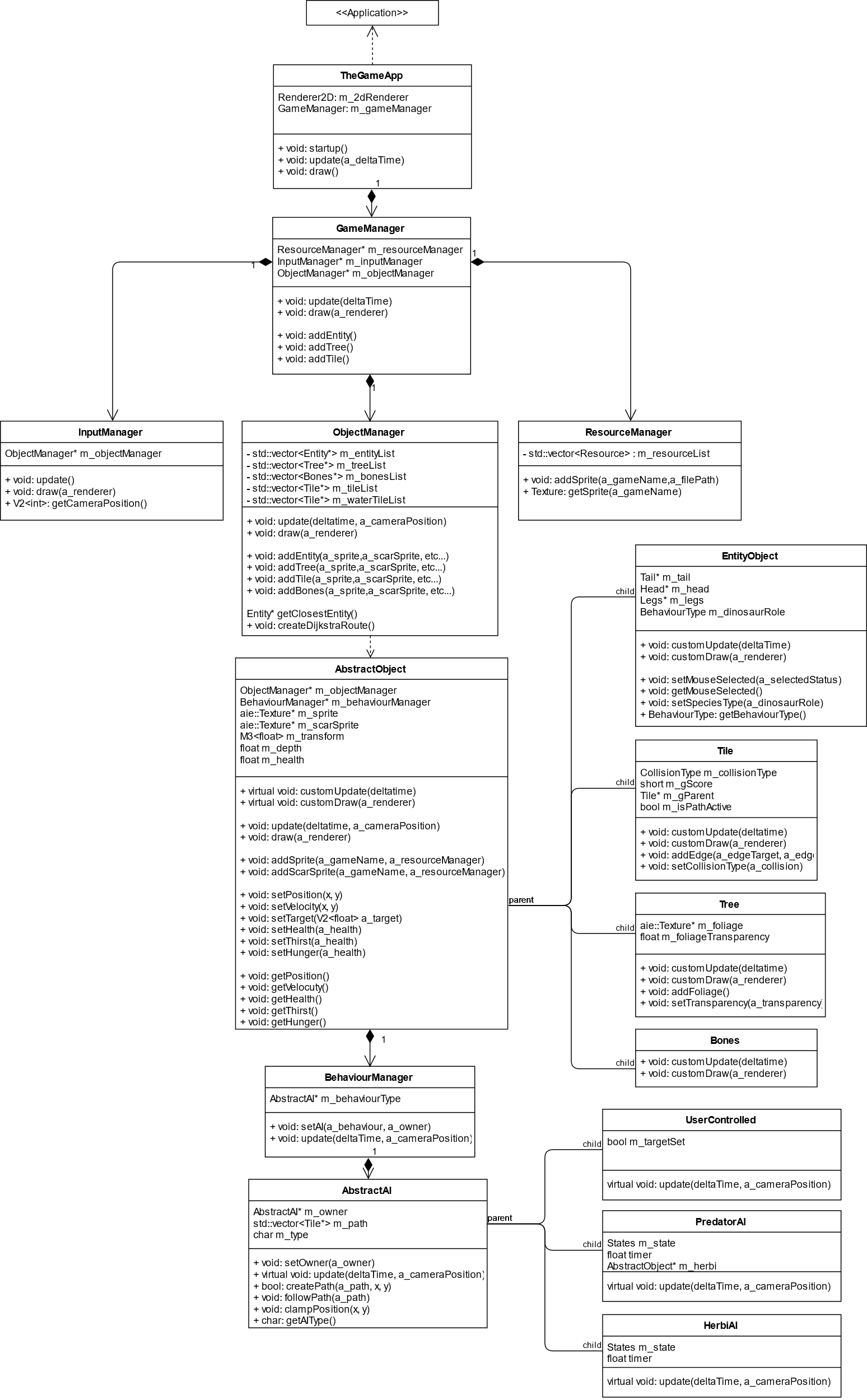
## Third Party Libraries

* CRT Library is used for checking memory leaks.
* Various c++ libraries such as vectors are used.

## Other Software

* Photoshop will be used to design the graphics.

# Game Overview

****

**GameManager**

The GameManger generates the three main managers, the **InputManager**, **ObjectManager** and **ResourceManager**. It also updates them during every step.

**ResourceManager**

The resource manager stores the sprites in a resource list with a short name that can be accessed by the GameObjects.

**ObjectManager**

The object manager stores and manages all the **GameObjects** and their derived classes on the map.

**InputManager**

The input manager manages the majority of the user input such as moving the camera and selecting multiple objects.

**AbstractObject**

The abstract object is the base class that represents all possible game objects such as dinosaurs (entities), tiles, bones etc. It includes a list of base functions used by all of its derived classes.

**BehaviourManager**

The BehaviourManager manages the behaviours associated with the objects. In general, the objects can be AI controlled (through finite state machines) or user controlled.

**Tile**

The Tile represents the non-interactive scenery found on the map.

**Entity**

The Entity object represents the dinosaurs, both predators and herbivores.

**Tile**

The Tile object represents the background tiles such as grass, rock or water. Rock tiles are solid and can’t be traversed.

**Tree**

The Tree object represents the trees visible on the map which can be eaten by herbivores. They change transparency if a dinosaur passes under them.

**Bones**

The bone object represents dead dinosaurs.

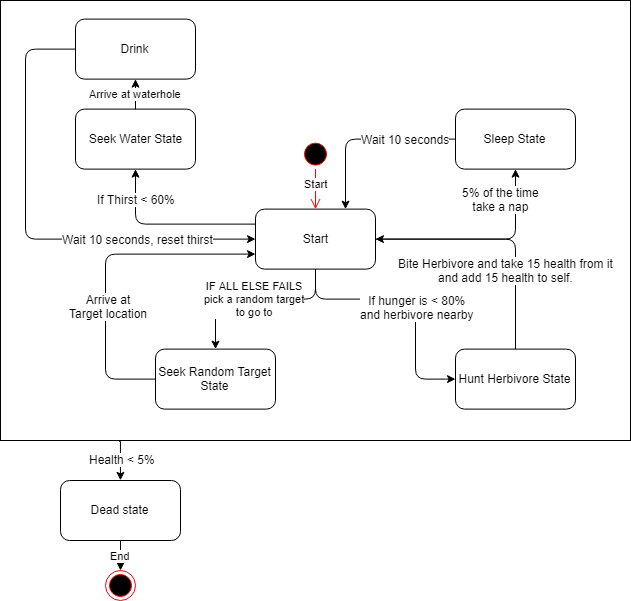
# Graphics

This will be a top-down birds-eye-view style game.

# Artificial Intelligence

#### **Predator AI**

The predator is represented graphically by a T-rex (it’s not a crocodile).

****

**Start State**

The predator picks a state based on the following paramaters.

* If Thirst is below 60% enter the **Seek Water State**. This uses Dijkstra’s Algorithm to find the nearest waterhole.
* If Hunger is below 80% and and a Herbivore is nearby enter the **Hunt Herbivore State**.
* If either of the above isn’t true, 5% of the time take a nap and enter the **Sleep State**.
* Otherwise pick a random point nearby and enter the **Seek Random Target state**.

**Seek Random Target State**

The predator picks a random **tile** on the map within 1000 pixels of itself and begins walking towards it. It uses **Dijkstra’s algorithm** to select the best path to reach the tile. I’ve choosen Dijkstra’s algorithm due to the fact that predators won’t need to navigate the entire map, therefore, the algorithm won’t be too resource intensive. If the predator reaches the tile it changes back to the **Start State**.

**Seek Water State**

The predator picks the closest waterhole nearby and begins walking towards it. It uses **Dijkstra’s algorithm** to select the best path to reach the tile. When it reaches the waterhole it changes to the **Drink state**.

**Drink State**

The predator sits still at the water hole for 10 seconds. After 10 seconds thirst is reset to 100% and it returns back to the **start state**.

**Hunt Herbivore State**

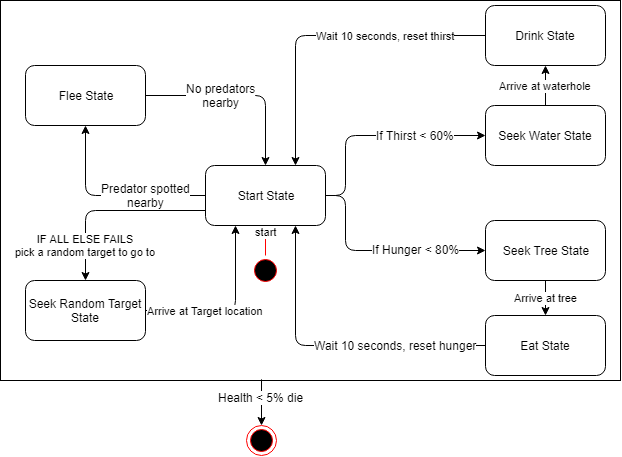
The predator chases the closest Herbivore nearby constantly attempting to hit it. Each hit reduces the Herbivore’s health by 15% and restores 15% of the predators Hunger. After each hit it returns to the **Start state**.

**Sleep State**

The predator sits still for 10 seconds. After 10 seconds it returns to the **start state**.

#### **Herbivore AI**

The herbivore is represented graphically by a long-necked blue dinosaur (it’s also not a crocodile).



**Start State**

The herbivore picks a state based on the following paramaters.

* If Thirst is below 60% enter the **Seek Water State**. This uses Dijkstra’s Algorithm to find the nearest waterhole.
* If Hunger is below 80% and enter the **Seek Tree State**.
* Otherwise pick a random point nearby and enter the **Seek Random Target state**.

**Seek Random Target State**

The herbivore picks a random **tile** on the map within 1000 pixels of itself and begins walking towards it. It uses **Dijkstra’s algorithm** to select the best path to reach the tile. I’ve choosen Dijkstra’s algorithm due to the fact that predators won’t need to navigate the entire map, therefore, the algorithm won’t be too resource intensive. If the herbivore reaches the tile it changes back to the **Start State**.

**Seek Water State**

The herbivore picks the closest waterhole nearby and begins walking towards it. It uses **Dijkstra’s algorithm** to select the best path to reach the tile. When it reaches the waterhole it changes to the **Drink state**.

**Drink State**

The herbivore sits still at the water hole for 10 seconds. After 10 seconds thirst is reset to 100% and it returns back to the **start state**.

**Flee State**

The herbivore flees from the closest predator nearby. After the predator is out of range it returns back to the **Start state**.

**Seek Tree State**

The herbivore picks the closest tree nearby and begins walking towards it. It uses **Dijkstra’s algorithm** to select the best path to reach the tile. When it reaches the tree it changes to the **Eat state**.

**Eat State**

The herbivore sits still at the tree for 10 seconds. After 10 seconds hunger is reset to 100% and it returns back to the **start state**.

# Physics

The GameManager will manage collision checking of all objects. It will use an Radius based Bounding Box and SOLID tiles.

# Items Item

There are no item that need to be collected.

# Game Flow

This is a simulator. The GameManager will randomly generate a map and place Predators and Herbivores (prey) on the map. The game will then continue to play until all Herbivores and/or Predators die..

# Levels

There are no levels to the game.

# Interface

The interface will be minimalistic. It will consist of a small health, hunger and thirst bar for selected entities.

## Menu

There is no menu.

## Camera

The camera can be panned LEFT-RIGHT-UP-DOWN ny using their respective keyboard keys.

## Controls

The selected entity is controlled by the mouse and keyboard.

# Asset List

The following graphics need to be created in Photoshop:

* Predator legs, body and head both healthy and wounded.
* Herbivore legs, body and head both healthy and wounded.
* Three sets of basic rock tiles to give variety.
* Water tile.
* Tree and foliage.

# Technical Risks

The only technical risk I forsee is running out of time.