Title Page

Flappy Turtle – My Own interpretation of Flappy Bird

# Document History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author(s) | Changes |
| 1.0 | Jan 2022 | D. Burchill |  |
| 1.1 | Feb 2022 | A.Nguyen | Added changes for flappy turtle game |
|  |  |  |  |

# Table of Contents

Table of Contents

[1 Document History 2](#_Toc96885382)

[2 Table of Contents 3](#_Toc96885383)

[3 Game Summary 4](#_Toc96885384)

[4 Development Environment 4](#_Toc96885385)

[4.1 Development Hardware 4](#_Toc96885386)

[4.2 Development Tools 4](#_Toc96885387)

[4.3 External Code 4](#_Toc96885388)

[4.4 Game Engine 4](#_Toc96885389)

[5 Architectural Analysis 4](#_Toc96885390)

[5.1 Classes 4](#_Toc96885391)

[5.2 Class Diagrams 6](#_Toc96885392)

[5.2.1 SceneNode 7](#_Toc96885393)

[5.2.2 SceneGraph Object diagram 8](#_Toc96885394)

[5.2.3 StateStack 9](#_Toc96885395)

[5.2.4 Application 10](#_Toc96885396)

[5.2.5 Turtle 11](#_Toc96885397)

[5.2.6 Game 12](#_Toc96885398)

[5.3 Behavioral Analysis 12](#_Toc96885399)

[5.3.1 Game States 12](#_Toc96885400)

[5.4 Game Loop 13](#_Toc96885401)

[6 Technical Risks 13](#_Toc96885402)

# Game Summary

Flappy Turtle is the side scroller game in which the player will try to survive by either not crashing into the moving pillars or the sharks that come and randomly on the screen and gain the most points possible. The game will get harder and harder as the player progresses.

# Development Environment

## Development Hardware

*Specify the primary programming language(s), including version (e.g. C++03), to be used for development. Also specify any scripting languages that are used (if applicable).*

Player control is through the keyboard and no special hardware has been used. This game and game engine have been developed in a MS Windows environment. It’s built using SFML 2.5 and C++17, so it could be built for Linux or MacOS platforms too. Programming Languages

## Development Tools

*List the software needed for development, such as IDEs, compilers, debuggers, version control software, graphical editors, sound editors, etc.*

* Microsoft Visual Studio and C++ compiler was used as the IDE and compiler.
* Git was used for Version control

## External Code

List the libraries to be used, including the URLs of the official library site, and the version number of the library used in this project.

SFML, the simple fast multimedia library was used for all graphics, <https://www.sfml-dev.org>. SFML is a simple cross platform wrapper around Open-GL providing a simple interface for graphics programing.

## Game Engine

The Gex game engine consists of a SceneGraph to hold all game objects and manage update and draw calls for all objects in the game, a group of State classes to represent different game scenes or game states. The sample game has only one scene or game state, along with a title state, a menu state, a pause state, and a game over state.

The sample game state was a world class in which the sceneGraph is populated, and collisions are handled.

# Architectural Analysis

## Classes

Describe the classes that will have to be implemented. For each class, provide:

* Its responsibilities
* How it collaborates with other classes

Note: Classes in purple are specific to the sample game, all other classes are game engine only.

SceneGraph:

The sceneGraph is a tree data structure that holds all of the game objects. It has layer nodes to control draw order, nodes attached to lower layer nodes are drawn first, meaning they are covered if nodes are later drawn at the same x,y location. All nodes in the sceneGraph are transformable and drawable.

|  |  |
| --- | --- |
| Class | Responsibilities |
| SceneNode : sf::transformable, sf::drawable | Manages the SceneGraph data structure. Each node manages the life cycle of it’s children nodes. To be in the sceneGraph a class must inherit from sceneNode. A sceneNode object is both sf::transformable and sd::drawable.  All scene nodes in the sceneGraph are updated and drawn in the gameLoop. |
| SpriteNode : SceneNode | Used, for putting in the sceneGraph , backgrounds and other such objects that need to be drawn, but have no physics and do not participate in collisions |
| TextNode : SceneNode | A sprite like object that displays text. |
| Entity : SceneNode | Game objects that have any velocity, hit points, or need to participate in collisions and physics simulations. |
| Turtle : Entity | Game specific, player |
| Pillar : Entity | Game specific, pillars |
| PillarGroup : Entity | Game specific, pillar groups |
| Sharks : Entity | Game specific, sharks |
| Ground : Entity | Game specific, ground element |
| Animation | Provides animation for Entities. Given a sprite sheet the frame is advanced to play the animation |
| Animation 2 | Same as animation, supports non regular sprite sheet |

|  |  |
| --- | --- |
| Class | Responsibilities |
| ResourceHolder | Template class to load and manage resources like textures, texture atlases, sound effects, fonts, etc. |
| PlayerControl | Object that receives and processes all player input, commands are created and put on the command queue to control the players avatar/character (player aircraft in sample game) |
| Command | Command object used to send commands to objects in sceneGraph |
| CommandQueue | Queue of commands that are sent to all nodes in the sceneGraph during the update cycle |
| PostEffect | Shader effects |
| BloomEffect : PostEffect | Bloom effect |
|  |  |
| Application | Initialize the game and runs the game loop |
|  |  |
|  |  |
| State | A game state/screen/scene |
| TitleState : State | Game Title or splash screen |
| MenuState : State | Game Menu |
| PauseState : State | Pause screen |
|  |  |
| GameState : State | The Game |
| World | The game world, all initial game objects are created and put in the sceneGraph, collisions are handled, and the sceneGraph is updated and drawn in the game loop. |
| StateStack | Holds and manages game states |
| Particle | A particle for use in a particle system, details are looked up in the data tables, |
| ParticleNode : SceneNode | A particle system, owns, updates and draws all particles in the particle system |
| EmitterNode: SceneNode | Emits particles into a particle system |

DataTables:

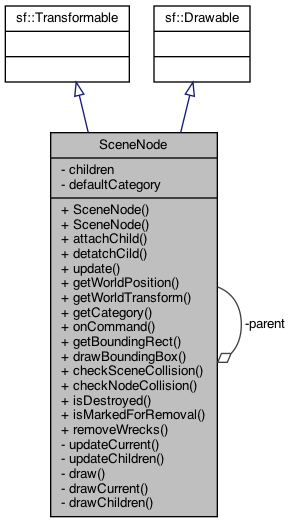
Look up tables are used for the sample game. The type of aircraft, projectile (bullet or missile), pickup, and particle system are defined and looked up from these tables.

|  |  |
| --- | --- |
| Class | Responsibilities |
| TurtleData | Data related for the Turtle (textures, hp, textureRect, flyAnimation) |
| PillarData | Data relatted for the Pillars (textures,color) |
| SharkData | Textures for Sharks |

## Class Diagrams

Present class and object diagrams that show the relationships between classes. Show only the most essential attributes and methods for each class. You may use Doxygen to create class diagrams

### SceneNode



### SceneGraph Object diagram

Diagram

Description automatically generated

### StateStack

Diagram

Description automatically generated

### Application

Diagram

Description automatically generated

### Turtle

Diagram, schematic

Description automatically generated

### Game

### 

Diagram, engineering drawing

Description automatically generated

## Behavioral Analysis

*Present statecharts, flow charts (activity diagrams), sequence diagrams, etc. that model complicated behavior. If your game has actors that implement a state machine, this would be the section where you’d present the statechart.*

### Game States

Graphical user interface, text, application

Description automatically generated

## Game Loop

*Describe, in order, the sequence of activities that happen during each game loop. You must document this even if you’ll be basing your game on the games developed in class.*

The game state instantiates and has a World object. The world object initializes the world, building the sceneGraph and adding to it all of the entities that ared need at the start of the game, including the player aircraft. The world updates all of the objects in the sceneGraph and then draws all the objects in the sceneGraph every cycle of the game loop. During the Worlds update, all collisions are detected and handled, all objects that have gone out of view and are no longer relevant to the game are destroyed. When the sceneGraph is updated all commands in the command queue are sent into the sceneGraph so that sceneNodes can receive and execute their commands from the command queue.

# Technical Risks

List all technical risks that could make it difficult or impossible to complete the game. Examples:

* Uncertainty on how to implement a certain feature
* Uncertainty on if a certain feature can be executed fast enough in real time
* First time using a certain library

|  |  |  |
| --- | --- | --- |
| Risk | Severity | Mitigation (what is to be done to eliminate or minimize this risk) |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |