Win criteria: All enemies reduced to 0 lives.

Loss criteria: Player reduced to 0 lives.

Continue criteria: Both player and enemies exceed 0 lives.

**MINIMUM VIABLE PRODUCT REQUIREMENTS**

Program class (called “Assessment 4”)

* Start the game.
* End the game.

Game class

* The scene for a play session
* Draw scene to screen
* Update scene calculations

Initialise class

* Set up initial conditions for a play session

Object class

* Default game object

Sprite class

* Hold a texture for a game object

Player class

* A derived class of Object for the player

Enemy class

* A derived class of Object for the enemy

Weapon class

* A class of for player and enemy weapon types

Controller class

* Calculate all game changes arising from player inputs

Maths class

* Perform all arithmetic for the controller, weapons and enemies

What should the program class do?

* Start the game
  + Instantiate a Game class
* End the game
  + Destroy the Game class
  + Run shutdown on program

What should the game class do?

* Run initialisation
  + Instantiate initialisation class
* Update the timer
* Update relationships
  + Add objects from the scene
    - Add new objects to a list to be added to the scene on next update
  + Remove objects from the scene
    - Remove new objects from the list of items to be added to the scene on next update
* Update calculations
  + Run Update function
    - Parents calculate themselves and their children
* Update the Draw
  + Scene UI draws itself
  + List of parents Draw themselves and their children
* Detect game end

What should the initialise do?

* Set the parameters for the game to run after starting and before ending
  + Set game-wide parameters
    - Create the visible play area
    - Create a timer
    - Create and set player score = 0
    - Set target FPS
* Set the win and lose conditions
  + Win if enemy lives = 0
  + Lose if player lives = 0
* Instantiate all of the objects
  + Instantiate the player object
    - Set initial player position
    - Offset object position to middle of object rather than top-left
      * Set size of the offset for collision detection?
    - Set lives = 1
    - Instantiate a player sprite child object
  + Instantiate enemy
    - Set initial enemy position
    - Offset object position to middle of object rather than top-left
      * Set size of the offset for collision detection?
    - Set lives = 1
    - Instantiate an enemy sprite child object
  + Instantiate the base
    - Set position
    - Lives = 1
    - Instantiate base sprite child objects

What should the object class do?

* Create a blueprint for any object with a transform in the game
  + Have a parent object (optional)
  + Have a list of children
  + Have a transform (vector of ints?)
    - 3D matrix for local transform
    - 3D matrix for global transform
  + Offer types (enum?)
    - Player
    - Enemy
    - Base
    - Weapon / projectile
  + Lives integer
  + Instantiate a weapon for this object
  + Functions for use by a controller / movement pattern
    - SetPosition
    - Translate
    - CopyTransform (set local to global)
  + Create virtual OnUpdate function for use by separate types
    - Player
    - Enemy
    - Base
    - Weapon / projectile
  + Draw this object and its children
    - Have override OnDraw function

Sprite class

* Load a texture
* Set the sprite’s height equal to its texture height
* Set the sprite’s width equal to its texture width
* Run OnDraw method
  + Draw the texture

What should the player class do?

* Be a container for all of the parameters that collectively represent the player
  + Set the object type to ‘player’
  + Parent a sprite object to hold a texture
  + Lives
    - Receive default from initialisation
    - Receive information from the weapon class of enemies
  + Override parent OnUpdate function
    - Transform
      * Receive default from initialisation
      * Receive information from the controller
  + Weapon class object
    - Receive default from object type
  + Instantiate a copy of the controller class

What should the enemy class do?

* Be a container for all of the parameters that collectively represent an enemy
  + Set the object type to ‘enemy’
  + Lives
    - Receive information from the weapon class of the player
  + Transform
    - Execute a pattern
  + Parent a sprite object to hold a texture

What should the weapon class do?

* Distinguish between whether it is a player or enemy weapon
  + Binary option like “is this the player’s weapon?” maybe
* Receive inputs from the controller class
  + Shoot
* Receive inputs from the enemy class for periodicity of instantiation
  + Options for pattern of attack
    - Predictable
    - Random
* Send outputs to the screen (draw)
  + Transform
  + Despawn
* Calculate updates
  + Transform

What should the controller class do?

* Everything to do with keystrokes
  + Movement
    - Send information to the player class
      * Move the transform of the player
  + Attacks
    - Send information to the weapon class of the player
      * Instantiate attacks for the player

Maths library class

* Perform all arithmetic for the controller, enemies, weapons
  + Trigonometry, array, vector and matrix calculations

Learning

# General

* There is no default stopwatch in C++, I have to build one or use #include <ctime>
* #include the <cmath> library for arithmetic

# Classes

* I immediately discovered I don’t know how to instantiate classes in C++, because the C# format doesn’t work the same.
* I need to remember that to instantiate another class (B) from a class (A) the referencing class (A) needs to #include the header file of the other class (B)
  + Assessment 3.cpp needs #include “Game.h”
  + ALWAYS CREATE INSTANCES AS POINTERS

# Methods

* I need to immediately define a constructor and destructor myself in the header file, and then implement their definition in the CPP file.
  + Eg in Header
    - Game();
    - ~Game();
  + In CPP
    - Game::Game(){}
    - Game::~Game(){}
* C++ constructors and destructors don’t have return types when being declared

Assessment 4 class