TECHNICAL DESIGN DOCUMENT: HAEMORRHOIDS!

A FANTASTIC VOYAGE

Devin Grant-Miles

COMP710 Game Programming

Version 1.0

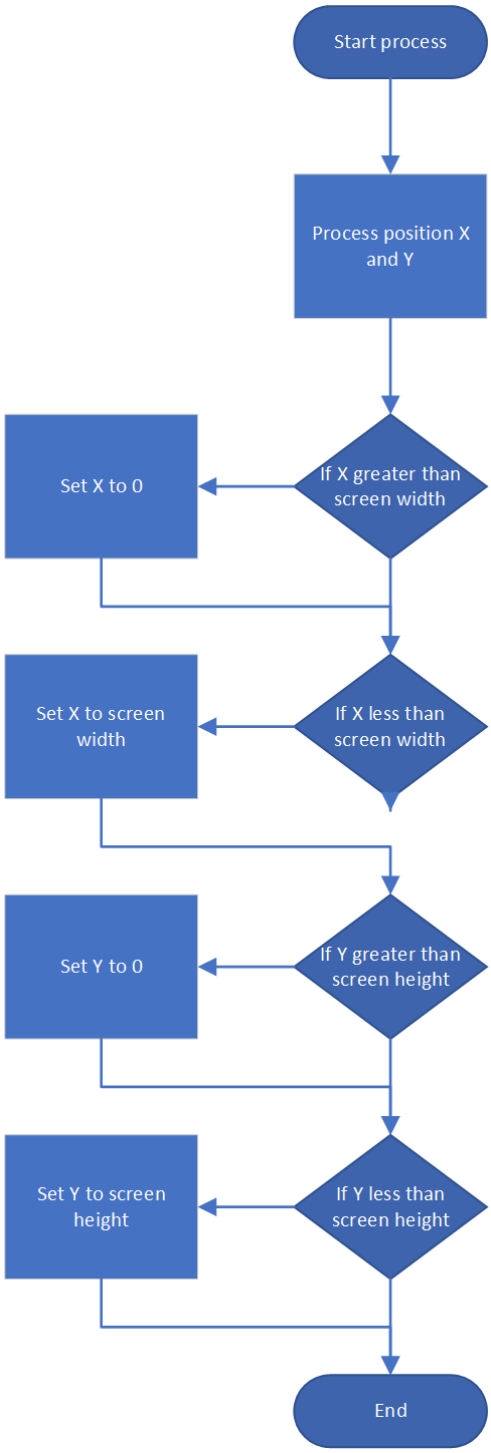
23 September 2020

Table of Contents

Key game logic and algorithms

Screen edge wrap algorithm

- Required for entity locations and drawing
- If an entity is outside of a screen edge, then its location is set to the opposite screen edge
- The algorithm will need to ensure that the draw method draws parts of objects according to the pixel's coordinate



Player ship thrust and rotation logic

Thrust should be implemented such that:

- The ship has a maximum speed possible
- The ship reaches a maximum within a couple of seconds
- Thrust button hold must be detected
- Thrust is converted to velocity in the thrust method using ship angle as the direction
- Deceleration friction may be applied each frame to increase playability.

Rotation should be implemented such that:

- Rotation speed is constant and is a factor of 360 so that the ship always returns to 0 degrees after a full rotation
- The ship should only rotate while stick is being pressed. Instant rotational acceleration and deceleration.
- Holding rotation analogue stick hold must be detected

Also note that the player is spawned in the centre of the screen. The player is reset to the centre of the screen at each collision, unless it is game over.

Player must not spawn while debris is within a certain radius of screen centre to avoid instant and/or unfair collisions.

Debris placement, movement and splitting/spawning logic

Debris placement should be as follows:

- Newly spawned debris must be given a start position.
    - Child debris will be spawned where destroyed parent debris was.
    - New level debris must be spawned outside of a certain perimeter of the centre; where the player ship will be spawned.
- A set number of debris should be spawned for each level.
- Debris may overlap with other debris – no collision detection.

Debris movement should be as follows:

- Newly spawned debris must be given a velocity and trajectory. This will not change for its lifecycle. Both will be random values; however, velocity must be set with certain bounds i.e. not too slow or too fast.
- Child debris may have slightly high velocity bounds than its parent.

Debris splitting should be as follows:

- Debris may only split in to two small debris that are half the size.
- Only large and medium sized debris split.

Score keeping and lives algorithm

Player lives logic:

- Player starts with three lives
- Player loses 1 life at each player ship collision
- Game over when lives reach zero
- Player reset to centre of the screen at each collision, unless it is game over
- Player must not spawn while debris is within a certain radius of screen centre

Score keeping logic:

- Haemorrhoid 1000pts
- Large debris 100pts
- Medium debris 200pts
- Small debris 300pts
- LERPing bacteria 500pts

Level logic

- At each increased level, difficulty is increased by increasing the number of starting large debris by 1 by and the upper bounds of debris speed increases by a certain factor (to be tested)
- Music should be changed at each level. Beyond level 5 it stays the same. Tempo related to debris speed.
- Level counter should be increased.
- There should be a maximum amount of debris allowed as well as a max debris speed.

Collisions logic

- Player ship and bullets cannot collide with each other, only debris and haemorrhoids.
- Therefore, only collision detection between these entities is required and can be managed entirely by the debris manager and haemorrhoid class.

Bullets management logic

- Playership has a BulletPool member which consists of a fixed array of bullet objects

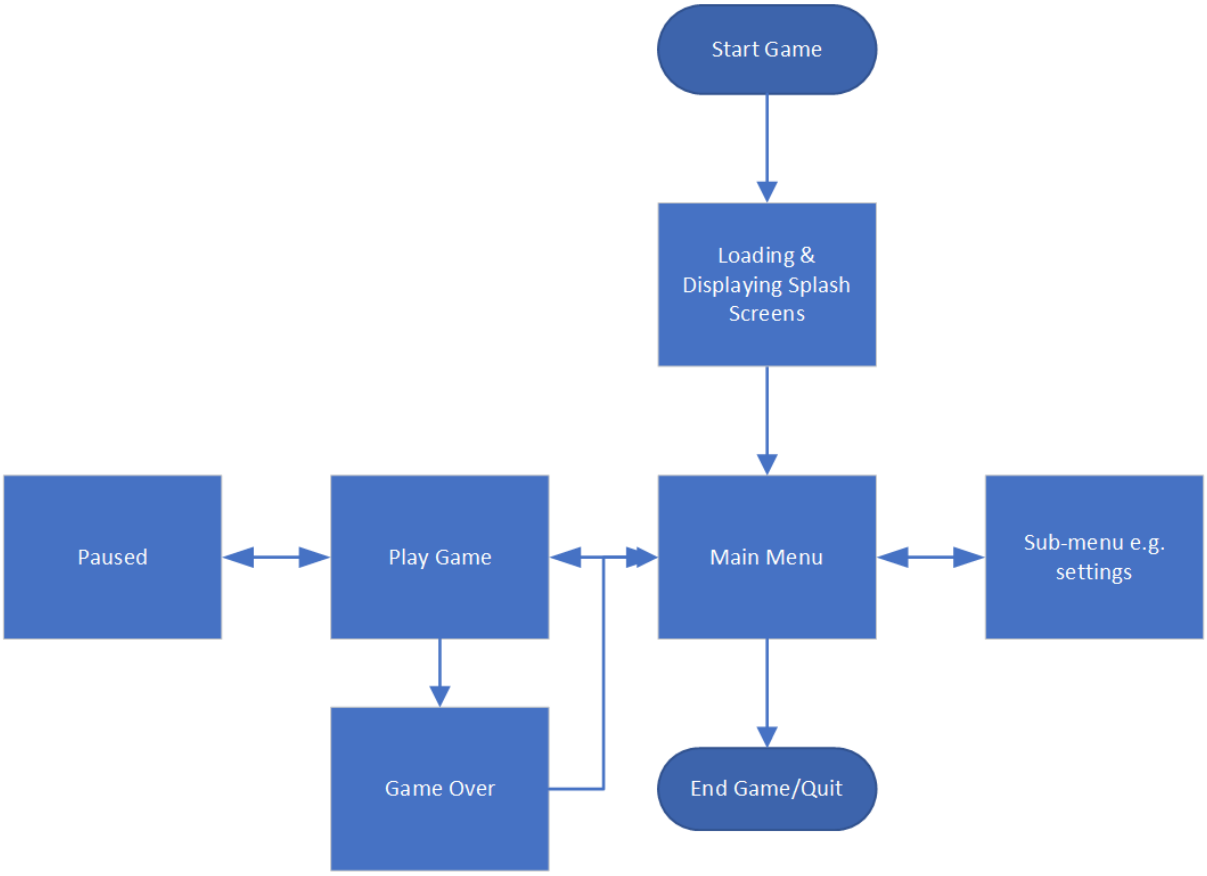
- When a bullet is fired it is set to isDead(false)
- Bullets have a maximum range. Once a bullet reaches its maximum range it is alpha blended out and set to isDead(true) for reuse.
- The array of bullets should be of sufficient quantity to cover a bullets entire lifecycle while being shot at maximum frequency
- Bullets must move at a velocity that is additional to the current player velocity.

Particle management algorithm

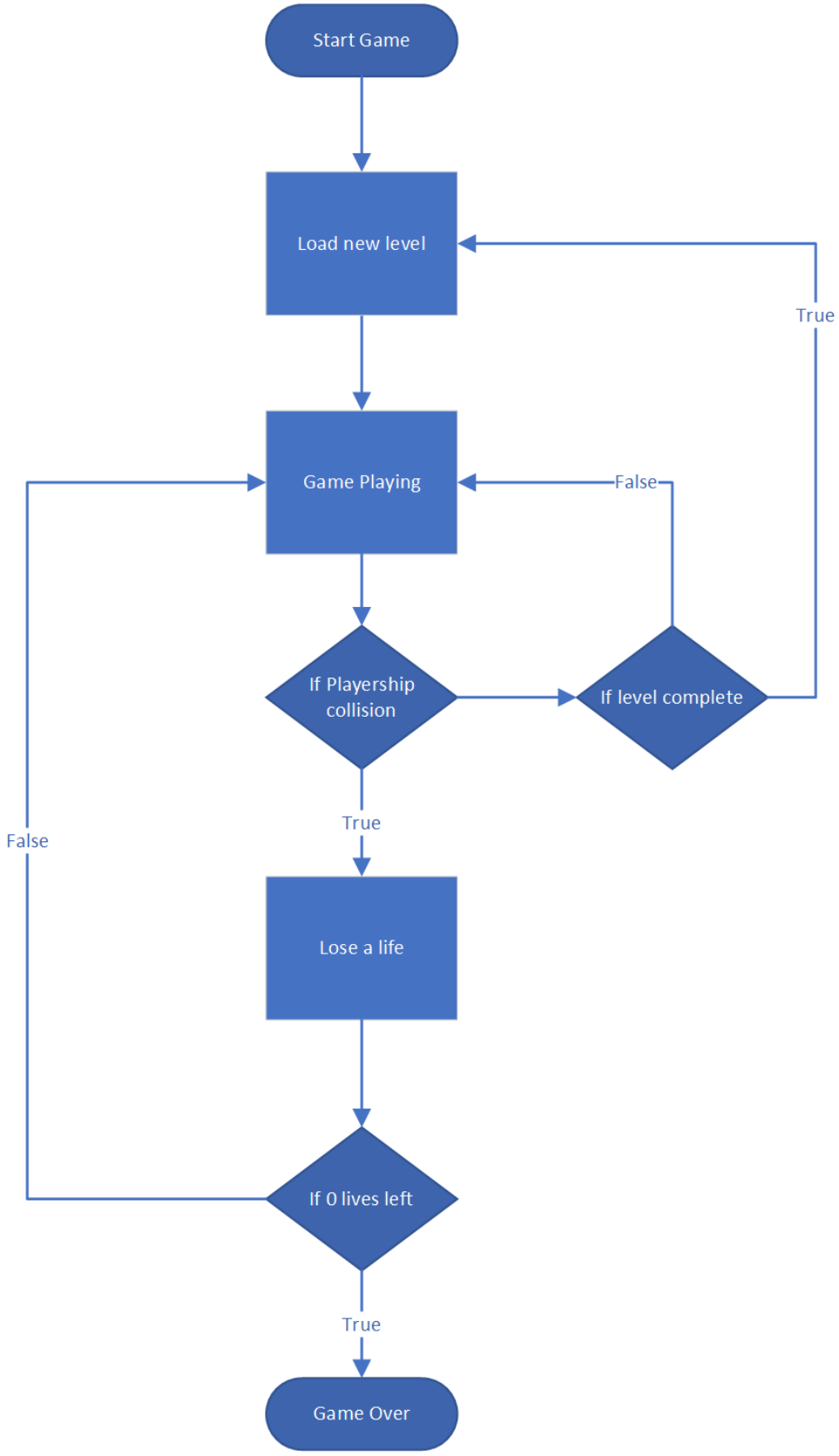
- Like bullets, a fixed number of particles created at start using object pools; arrays of particles managed and held within the particle emitter subclass.
- Particles to be alpha blended out over time. Once at max transparency life particles are set to dead and reused.
- Particles are all explosion type particles so to be set at random trajectories and random velocities (within a particular range)

Key game loops

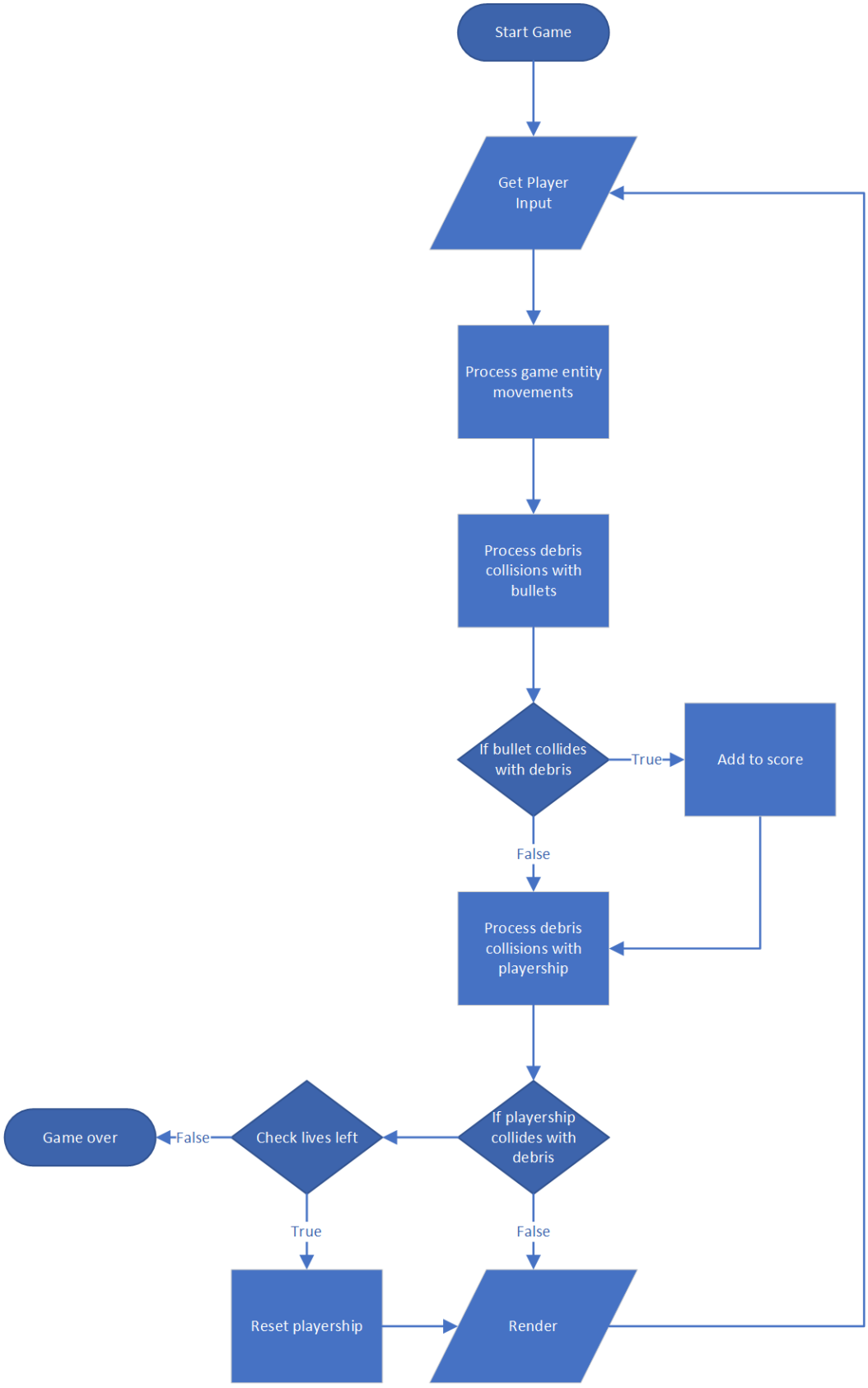
Overall game loop – menu included



Basic overall in-game loop



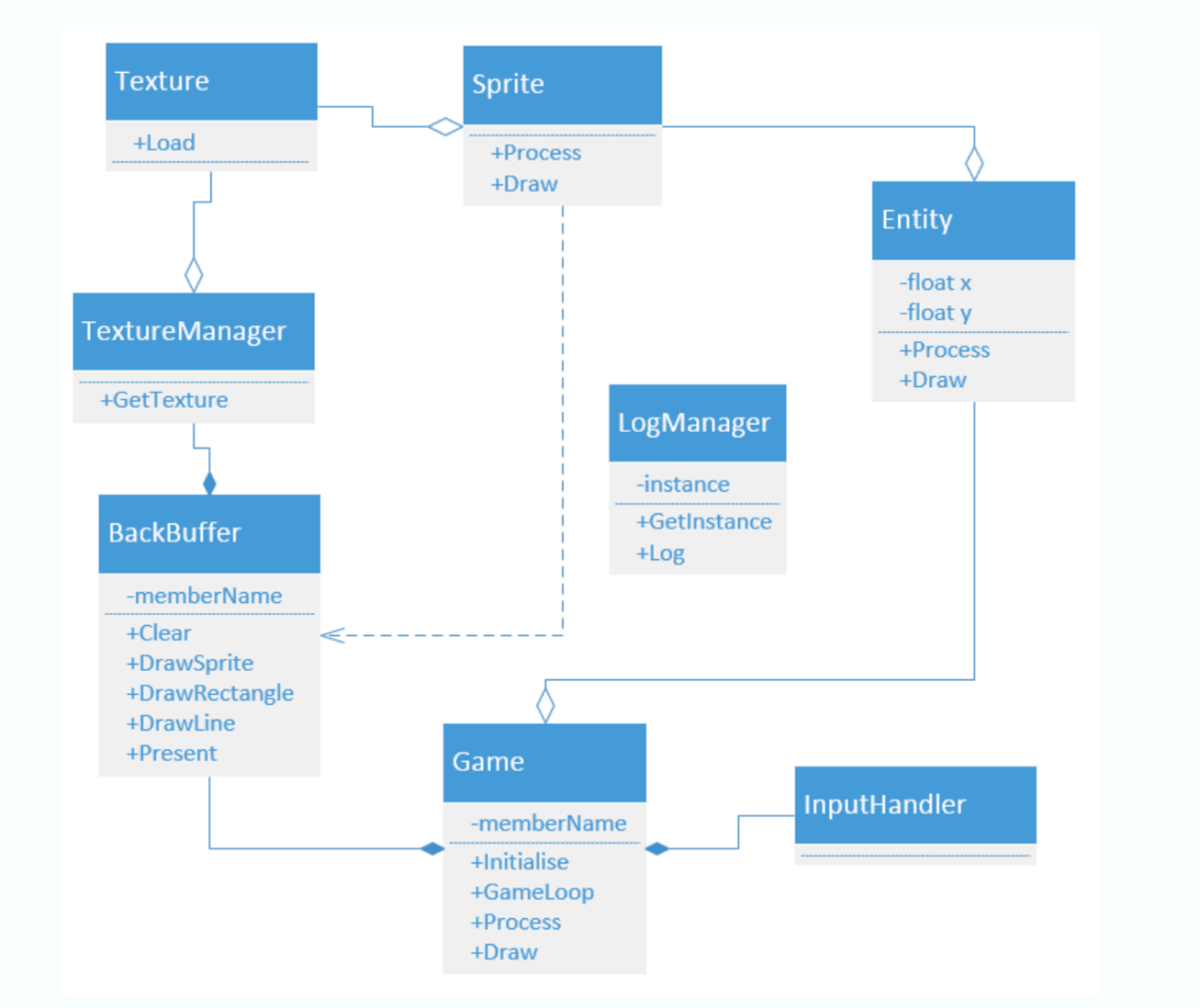
Real-time in-game loop



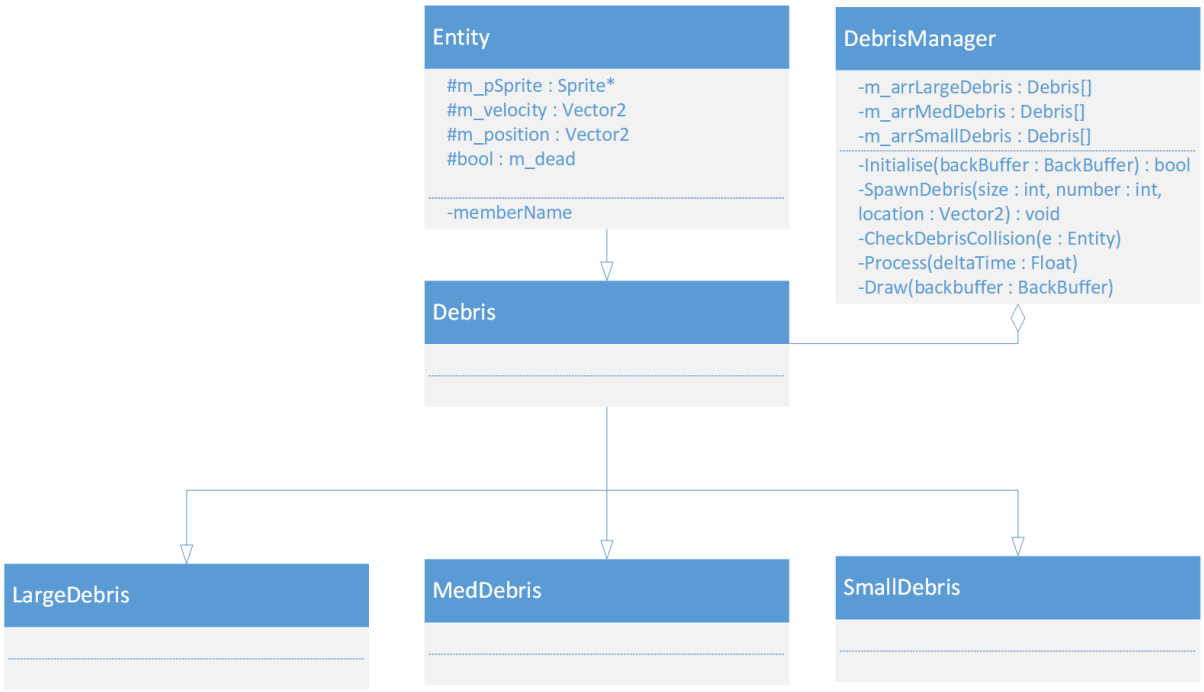
Key game objects, hierarchies and relationships

Basic game framework

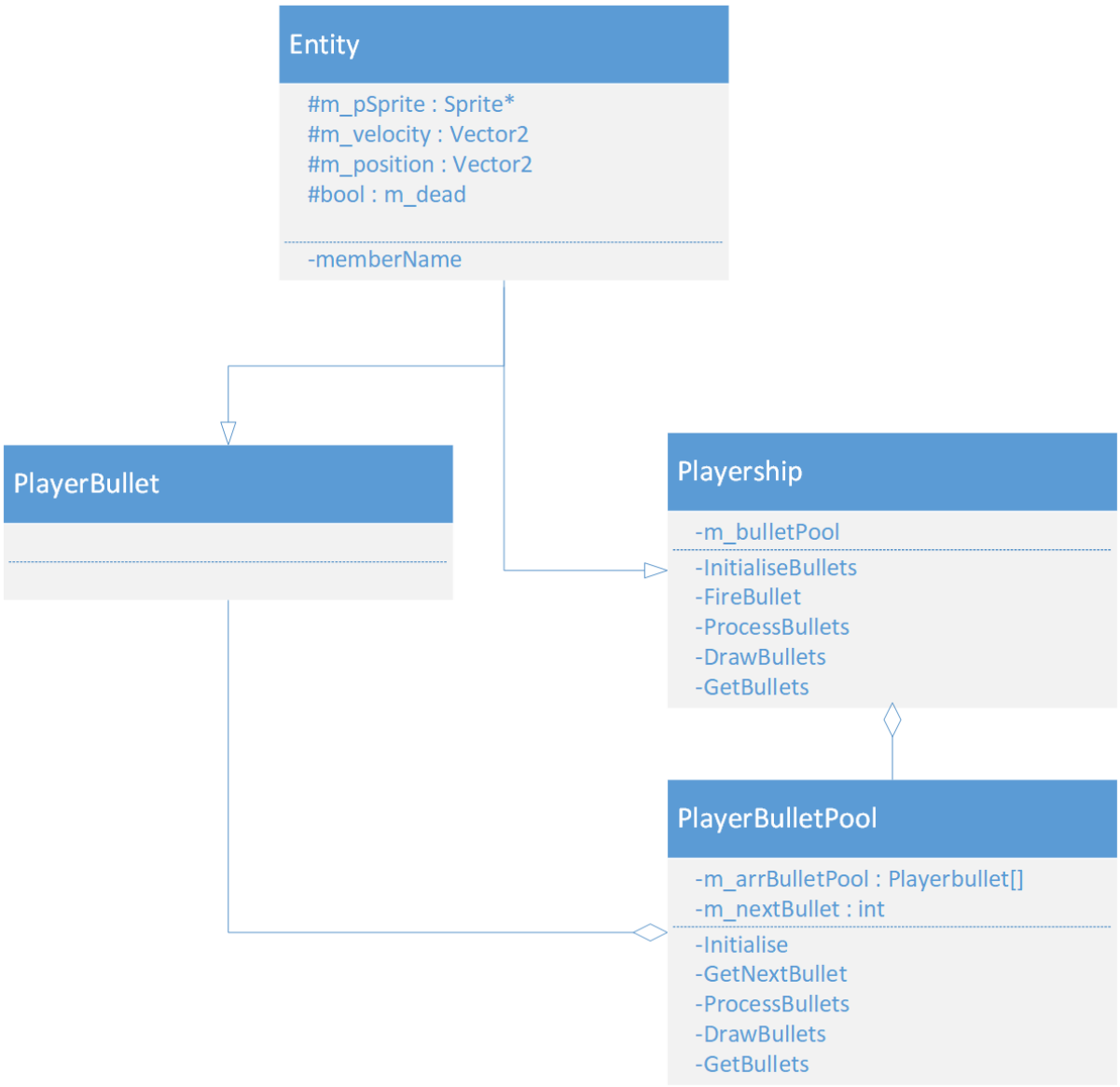
The basic framework which forms the basis of the following UML diagrams (Hooper, 2020b).



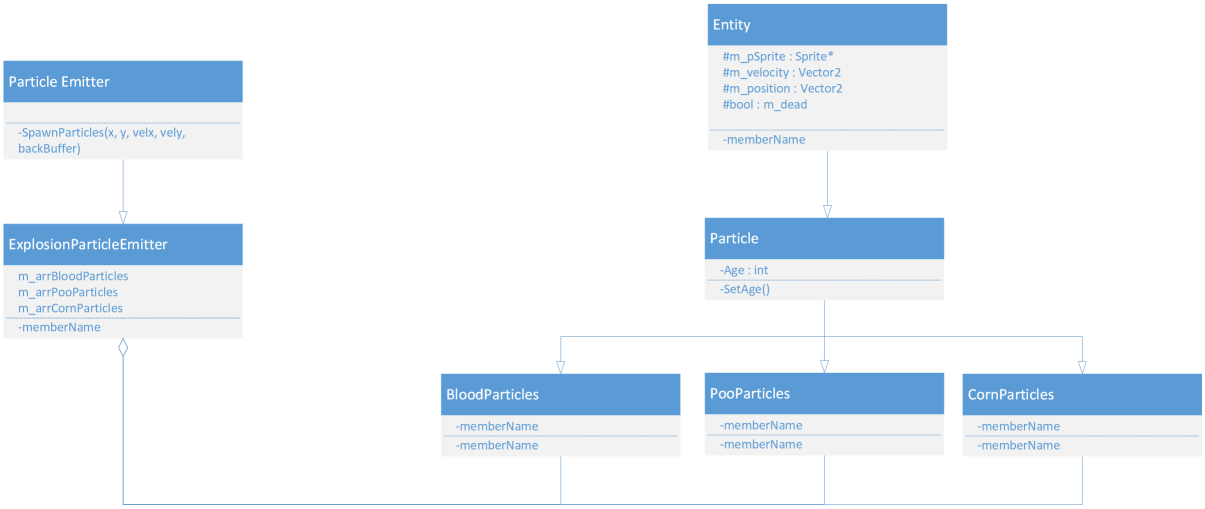
Basic Debris object hierarchy



Playership object hierarchy



Particles/particle management object hierarchy



Debug features

The game should feature the following debug tools

- A player ship invincibility option so the tester can progress through all levels without having to start over
- An option to allow level selection so that specific levels can be tested individually
- Pause functionality so that testing can be paused and specific frames can be reviewed.

The pause functionality may be included in the final product for player use.

Development standards

Coding standards and naming schemes

Use the following rules and guidelines during development. The overall goal is an aesthetically pleasing style that is easy to interpret, even if at the cost of verbosity.

Naming and stylistic standards

- Uppercase PascalCase for:
  - Type names (struct, class, enum)
  - Methods
- Lowercase pascalCase for:
  - Variable identifiers
- No leading or trailing underscores
- Member data start with m_ (e.g. m_variable)
- Pointers start with p_
- Member pointers start with m_p
- Pointers – group asterisk with the type, not the variable
- No using the comma operator to declare multiple variables on the same line
- Describing identifiers – long is OK
- Numerals OK in identifier names, but not leading
- Source code files all lowercase naming
- When choosing a name, prefer long names to abbreviated names. Prefer descriptive names to arbitrary ones

Key coding standards

- Return types of functions should have their own line in .cpp file

- Single return value in functions
- Do not use the using keyword - all namespaces explicitly referenced
- Tab indents, not double space
- Always use scope brackets and use separated line
- NZ spelling
- Accessors = set, get, is, does
- Do not use #include if a forward declaration would do

Best practice

- Regular and descriptive commenting
- Prune commented-out and unused code
- Avoid negative flags
- Whitespace
- No magic numbers

Relevant file formats

- Sound files must be in wav format
- Art, animations, sprites must be in PNG format
- C++ source files (.cpp) and standard C/C++ headers (.h)
- Microsoft Visual Studio .NET solution (.sln).

Middleware and libraries

- COMP710 2D C++ Framework
- C++ Standard Template Library (STL)
- Simple DirectMedia Layer (SDL2) including SDL2_Image and SDL2_TTF APIs;
- FMOD Core (Low-Level) API.

Object Pools

The following game objects will be stored in arrays and generated at the beginning of the game.

- Debris
- Bullets
- Particles

Particles and bullets will be alpha blended before they are reused.

Data driven design

Should be used by way of .ini file and should control velocities, health, quantity of entities.

Acceptance plan

1. Assets wrap around screen. Partially off-screen assets are drawn on both sides of screen.
2. Game includes a player ship which can be controlled by the player to thrust, shoot, and rotate. Its movement is smooth, responsive and fun.
3. Game includes all listed debris (including haemorrhoid) which destroys, splits and moves as prescribed in this document.
4. Game loads near instantly and runs smoothly.
5. Game levels can be won and scores accrue such that game can be won via a high score.
6. Lives are lost upon player ship/debris collisions and running out of lives ends the game.
7. Game loads showing splash screens, start animations and loads menu which functions as described in the documents. Can be started over from this menu without restarting application.
8. Game includes all HUD elements as described in this document and GDD.
9. Game plays with both keyboard and controller.
10. Game includes all sound effects, particle effects and animations as described in this document.

Bibliography

Asteroids (video game). (2020, September 6). In *Wikipedia*. https://en.wikipedia.org/wiki/Asteroids_(video_game)

Hooper, S. (2020a). *COMP710 Game Programming: W01 Studio Session 02 Lec 02a - C++, Classes, OOP, STL, Coding Standards* [PowerPoint slides]. Blackboard. https://blackboard.aut.ac.nz/

Hooper, S. (2020b). *COMP710 Game Programming: W02 Studio Session 04 Lec 04a - Real-Time Simulation, 2D Graphics, GP 2D Framework* [PowerPoint slides]. Blackboard. https://blackboard.aut.ac.nz/

Hooper, S. (2020c). *COMP710 Game Programming: W02 Studio Session 04 Lec 04b - Graphical Effects, Vector Mathematics* [PowerPoint slides]. Blackboard. https://blackboard.aut.ac.nz/

Hooper, S. (2020d). *COMP710 Game Programming: W05 Studio Session 07 Lec 07a - Data Structures for Games* [PowerPoint slides]. Blackboard. https://blackboard.aut.ac.nz/

Hooper, S. (2020e). *COMP710 Game Programming: W05 Studio Session 07 Lec 07b - Data Driven*

Design, Scripting, Tools [PowerPoint slides]. Blackboard. https://blackboard.aut.ac.nz/

Hooper, S. (2020f). *COMP710 Game Programming: W05 Studio Session 08 Lec 08b - Game Production*

[PowerPoint slides]. Blackboard. https://blackboard.aut.ac.nz/

Insomniac Games. (2011, October 3). *Core Coding Standards.*

https://gist.github.com/Kerollmops/fcad27cfef9e3552cb75a3d201494ba6

javidx9. (2020, June 13). *Essential Mathematics For Aspiring Game Developers* [Video]. YouTube.

https://www.youtube.com/watch?v=DPfxjQ6sqrc

Masó, V. (2018). *2D Particle System: A tutorial on how to implement a 2D particle system in computer graphics.* https://nintervik.github.io/2D-Particle-System/

Nuclear Monkey Software. (2004). Narbacular Drop: Technical Design Document. DigiPen (USA)

Corporation.

http://www.nuclearmonkeysoftware.com/documents/narbacular_drop_technical_design_document.pdf

Roast, K. (2014, September 15). *Asteroids [Reloaded]* [Website Video Game]. Kevin Roast. http://www.kevs3d.co.uk/dev/asteroids/index-debug.html