SD1 Assignment 2: Starship Playable

# Overview

Implement a simple “first playable” experience for the Starship game, building upon the foundational work for that project completed for Assignment 1.

For this assignment, we expect to see (roughly speaking):

* Add the ErrorWarningAssert.cpp/hpp and StringUtils.cpp/hpp provided in Canvas to your Engine/Core
* Game.hpp defines class Game; App owns Game; Game owns every Entity (PlayerShip, Asteroid, Bullet)
* Each Entity holds a pointer to the Game (Game\* m\_game), which is given to the Entity constructor
* Six Asteroids initially spawn at random positions, and tumble (spin) while drifting at a fixed speed
* PlayerShip, Asteroids, and Bullets are similar in size and appearance to those in the demo
* PlayerShip controls identical to demo (left/right arrows turn, up arrow thrusts, space fires, N respawns)
* PlayerShip physics identical to demo (acceleration, linear momentum, no angular momentum, etc.)
* Pressing space bar fires a bullet in the ship’s forward direction, flying out of “nose” at a fixed speed
* Asteroids and Bullets expire (and are deleted) if they move entirely offscreen (using cosmetic radius)
* PlayerShip “bounces” off the edges of the screen (i.e. reverses either its x or y velocity)
* If an Asteroid overlaps a Bullet or PlayerShip, the Asteroid loses 1 health and the other Entity dies
* Asteroids have 3 health; after being reduced to 0 (or less), they die
* Dead Entities have no gameplay interaction (e.g. can’t move, collide, turn, thrust, shoot, etc.)
* PlayerShip respawns (centered, facing east, with no momentum) if the player presses ‘N’ while dead
* Pressing ‘O’ spawns a new Asteroid in a random position; if there are already 12 Asteroids alive, open an ERROR\_RECOVERABLE dialogue and refuse to spawn the 13th Asteroid
* Likewise, only 20 living Bullets are allowed at once; open a dialogue and refuse to spawn a 21st Bullet
* Entities are not deleted immediately; rather, they are marked as garbage (m\_isGarbage = true); all garbage entities are deleted in Game::DeleteGarbageEntities(), called at the end of Game::Update()
* Pressing F1 toggles “developer mode”, in which each entity renders its inner & outer radii, velocity, etc.
* Pressing F8 hard-restarts the game (e.g. deletes m\_theGame and re-instantiates a new one)

*For SD1-A3 (Starship Gold), we will allow – and reward – more creative individual enhancements and “juice”.*

# Restrictions and Requirements

For this project (Starship – Assignments 1, 2, 3), the use of external code or libraries (e.g. Boost, Dear ImGui, TinyXML) as well as use of the **Standard Template Library (STL) is not allowed** – nor are smart pointers, templates, lambdas, or the “auto” keyword. You will therefore need to manage your entities (Ship, Bullets, etc.) using pointers and c-style arrays. Exception: you may use std::string, especially for debugging (e.g. with the Stringf() function provided in StringUtils.cpp/hpp). Note that in our second project, Incursion (Assignments 4, 5, 6) we will be allowed (and required!) to use STL containers such as std::vector.

## Requirements

1. ReadMe.txt (5 points)
   1. (2 points) A ReadMe.txt file in the Starship folder (alongside Starship.sln), added/submitted to P4; this should contain a clear, succinct description of anything I need to know in order to build, run, and play your game, including controls, known bugs, etc.
   2. (3 points) A Deep Learning metacognition/introspection section at the bottom of your ReadMe.txt. See the **How to write ReadMe Comments** document in Canvas for details and expectations.
2. App (4 points)
   1. (1) Handles app-level actions (such as the general frame, quitting, etc.)
   2. (1) Is owned by Main\_Windows.cpp
   3. (1) Owns all engine systems (for now, just the RenderContext, globally visible as g\_theRenderer)
   4. (1) Owns the Game instance (Game\* m\_theGame).
3. Game (5 points)
   1. (1) Gameplay overseer, bookkeeper, and referee – owns all gameplay-related concepts
   2. (1) Handles all entity-vs-entity interactions (e.g. physics, damage)
   3. (1) Handles all high-level game mechanics (e.g. levels/waves, spawning)
   4. (1) Is owned by g\_theApp
   5. (1) Owns all gameplay Entity instances (e.g. m\_playerShip, m\_bullets, m\_asteroids)
4. Entity (10 points)
   1. (1) Is an abstract base class (ABC), i.e. is never directly instantiated (e.g. never type “new Entity”)
   2. (1) PlayerShip, Bullet, and Asteroid all derive (inherit) from Entity
   3. (1) Provides base virtual implementations of each basic action (Update, Render, Die)
   4. (1) Constructor takes a pointer to the Game instance, and a starting position
   5. (2) Provides common inquiries/accessors (IsOffscreen, GetForwardVector, IsAlive, etc.)
   6. (4) Holds several universal data members used by most/all entities, including (at minimum):
      1. m\_position: the Entity’s 2D (x,y) Cartesian origin/center location, in world space
      2. m\_velocity: the Entity’s linear 2D (x,y) velocity, in world units per second
      3. m\_orientationDegrees: the Entity’s forward-facing direction, as an angle in degrees
      4. m\_angularVelocity: the Entity’s signed angular velocity (spin rate), in degrees per second
      5. m\_physicsRadius: the Entity’s (inner, conservative) disc-radius for all physics purposes
      6. m\_cosmeticRadius: the Entity’s (outer, liberal) disc-radius that fully encompasses it
      7. m\_health (int): how many “hits” the entity can sustain before dying
      8. m\_isDead: whether the Entity is “dead” in the game; affects entity and game logic
      9. m\_isGarbage: whether the Entity should be deleted at the end of Game::Update()
      10. m\_game: a pointer back to the Game instance
5. PlayerShip (28 points)
   1. (1) Derives (inherits) from Entity (e.g. class PlayerShip : public Entity)
   2. (1) The only player-controlled Entity
   3. (3) Turns (at 300 degrees per second) while Left or Right Arrow key is held (but not if both held)
   4. (4) Accelerates (30 world units/second/second) in its forward direction while Up Arrow key is held
   5. (2) Fires a bullet (at its “nose”, facing/moving in its forward direction) when space is pressed
   6. (3) Dies (but does not get deleted!) if it touches an Asteroid
   7. (2) Respawns (at screen center, facing right/east, with zero velocity) if N is pressed while dead
   8. (1) Has a physics radius of 1.75 (used for all collision detections), and a cosmetic radius of 2.25
   9. (2) Bounces off screen/world edges; cosmetic disc snapped in-bounds, x (or y) velocity is reversed
   10. (2) Is rendered as a vertex array of 5 triangles (15 vertexes); see Appendix A
   11. (2) Is drawn correctly rotated, according to the ship’s m\_orientationDegrees
   12. (2) Vertex array is constructed in local space (i.e. unrotated and centered around 0,0)
   13. (1) A TransformVertexArray() vertex utility function rotates & translates the ship before drawing:

TransformVertexArray( NUM\_SHIP\_VERTS, shipVerts, 1.f, m\_orientationDegrees, m\_position );

* 1. (2) The RenderContext’s DrawVertexArray() function is used to draw the ship in a single call:

g\_theRenderer->DrawVertexArray( NUM\_SHIP\_VERTS, shipVerts );

1. Bullet (13 points)
   1. (2) Spawned at the PlayerShip’s nose whenever space bar is pressed, facing & moving ship-forward
   2. (1) Moves forward (drifts) in a straight line (at 50 world units/second) until it dies
   3. (1) Faces the direction in which it moves (e.g. the long “tail” points backwards)
   4. (1) Has a physics radius of 0.5 (used for all collision detections), and a cosmetic radius of 2.0
   5. (1) Is rendered as a vertex array of 2 triangles (6 vertexes); see Appendix A
   6. (2) Rendered in the same fashion as the PlayerShip (vertexes stored as a local-space vertex array, use TransformVertexArray to rotate & translate, DrawVertexArray to render)
   7. (2) Dies if its m\_cosmetic radius disc goes entirely offscreen
   8. (2) Dies if it overlaps an Asteroid (using each Entity’s m\_physicsRadius disc)
   9. (1) Maximum of 20 Bullets alive at once; attempting to spawn an 21st Bullet opens an ERROR\_RECOVERABLE dialogue and refuses to spawn (note: should be difficult to achieve at normal speed, but easier in slow-motion)
2. Asteroid (21 points)
   1. (1) 6 Asteroids are spawned at start of a new Game (also, see the ‘A’ debug cheat key, below)
   2. (2) Spawned at a random position onscreen
   3. (3) Spawned with random orientation & random velocity (but with fixed speed = 10 units/second)
   4. (1) Moves forward (drifts) in a straight line (at 10 world units/second) until it dies
   5. (3) Spins while it drifts, at a constant angular velocity, chosen randomly within [-200,200] at birth
   6. (1) Has a physics radius of 1.6 (used for all collision detections), and a cosmetic radius of 2.0
   7. (2) Is rendered as a vertex array of 16 triangles (48 vertexes; see Appendix A)
      1. Each asteroid’s shape is randomly generated and unique; the “radius” of each vertex is randomly chosen between the asteroid’s inner (physics) radius and outer (cosmetic) radius.
      2. Each asteroid retains its random-generated unique shape once created (i.e. don’t re-randomize vertex radii every frame!)
   8. (2) Rendered in the same fashion as the PlayerShip (vertexes stored as a local-space vertex array, use TransformVertexArray to rotate & translate, DrawVertexArray to render)
   9. (1) Takes 1 damage if it overlaps a Bullet or Player (using each Entity’s m\_physicsRadius disc)
   10. (2) Dies if it takes 3+ points of damage
   11. (1) Dies if its m\_cosmetic radius disc goes entirely offscreen
   12. (2) Maximum of 12 Asteroids alive at once; attempting to spawn a 13th Asteroid opens an ERROR\_RECOVERABLE dialogue and refuses to spawn
3. Debug/Developer Cheats (9 points)
   1. (2) Pressing ‘O’ spawns another random Asteroid or opens an error dialogue (see above)
   2. (3) Pressing F1 toggles debug drawing for all entities;
      1. Each Entity’s cosmetic (outer) radius is drawn in magenta (255,0,255), and its physics (inner) radius is drawn in cyan (0,255,255)
      2. Each Entity’s current m\_velocity vector is drawn as a yellow (255,255,0) line segment
      3. A dark grey (50,50,50) line is drawn to the PlayerShip from the center of each other entity
   3. (2) Pressing F8 hard-resets the game (the App deletes and re-creates m\_theGame from scratch)
   4. (1) Pressing P toggles the game to pause/unpause (same as SD1-A1)
   5. (1) Holding T slows the rate of time to 10% normal (same as SD1-A1), while held
4. Miscellaneous (5 points)
   1. (1) The title of the window should be updated (for example, “SD1-A2: Starship Playable”)
   2. (1) The game achieves a fairly consistent framerate at about 60 Hz (60 frames per second).
   3. (1) Update() functions are passed deltaSeconds of ~0.0167 (1/60th second) unless slowed/paused
   4. (2) All Render() functions are const methods; that is, they do not change any game state

# Appendix A: Vertex Array layouts for various Entities

Vertex layouts for each of the three entity types. The dotted magenta circles indicate outer (cosmetic) radii; dotted cyan circles indicate inner (physics) radii.

|  |  |  |
| --- | --- | --- |
| D:\Root\sqp4\Projects\Guildhall\Courses\SD1\C28 - 2018 Fall\Uploads\Images\SD1-A2_StarshipVertexArray.png | D:\Root\sqp4\Projects\Guildhall\Courses\SD1\C28 - 2018 Fall\Uploads\Images\SD1-A2_BulletVertexArray.png | D:\Root\sqp4\Projects\Guildhall\Courses\SD1\C28 - 2018 Fall\Uploads\Images\SD1-A2_AsteroidVertexArray2.png |
| **PlayerShip**: 5 triangles (15 verts) | **Bullet**: 2 triangles (6 verts) | **Asteroid**: 16 triangles (48 verts)  Created with (cos,sin) in for-loop |

Vertex colors for PlayerShip are currently a “Bali Hai” teal-grey: Rgba8( 132, 156, 169, 255 ).

Vertex colors for Asteroids are currently a medium-dark grey: Rgba8( 100, 100, 100, 255 );

Vertex colors for Bullet are currently opaque yellow (255, 255, 0, 255) for the front tip, opaque red (255, 0, 0, 255) for the two side vertexes, and transparent red (255,0,0,0) for the tail vertex. Note this means the tail fades out toward the end of the tail.

# Appendix B: Gameplay constants (reference)

The following gameplay constants (or their equivalents) should be defined (perhaps in GameCommon.hpp):

constexpr int NUM\_STARTING\_ASTEROIDS = 6;

constexpr int MAX\_ASTEROIDS = 12;

constexpr int MAX\_BULLETS = 20;

constexpr float WORLD\_SIZE\_X = 200.f;

constexpr float WORLD\_SIZE\_Y = 100.f;

constexpr float WORLD\_CENTER\_X = WORLD\_SIZE\_X / 2.f;

constexpr float WORLD\_CENTER\_Y = WORLD\_SIZE\_Y / 2.f;

constexpr float ASTEROID\_SPEED = 10.f;

constexpr float ASTEROID\_PHYSICS\_RADIUS = 1.6f;

constexpr float ASTEROID\_COSMETIC\_RADIUS = 2.0f;

constexpr float BULLET\_LIFETIME\_SECONDS = 2.0f;

constexpr float BULLET\_SPEED = 50.f;

constexpr float BULLET\_PHYSICS\_RADIUS = 0.5f;

constexpr float BULLET\_COSMETIC\_RADIUS = 2.0f;

constexpr float PLAYER\_SHIP\_ACCELERATION = 30.f;

constexpr float PLAYER\_SHIP\_TURN\_SPEED = 300.f;

constexpr float PLAYER\_SHIP\_PHYSICS\_RADIUS = 1.75f;

constexpr float PLAYER\_SHIP\_COSMETIC\_RADIUS = 2.25f;

# Submission

Submit your assignment by following the instructions above and checking in all the required files to Perforce (including a Release-built Starship\_x64.exe), with the check-in comment “SD1-A2: COMPLETE” for the changelist you want me to grade. My Perforce changelist # **156984** *(replace “???” here; see instructions below)*

Also, in Canvas, you should submit a .zip file as follows:

* Submit a single .zip file to Canvas under the assignment.
* Your .zip should be named: **C29\_SD1\_A2\_LastnameFirstname.zip**

*For example, Jane Smith would submit a file named* ***C29\_SD1\_A2\_SmithJane.zip***

* Your assignment submission .zip file should contain the following:
  1. A **video recording** of you playing your game (and showing your code, as needed); be sure to visually demonstrate and verbally narrate each feature you want credit for
     + The video should be: 1920x1080 **.mp4** at 60 FPS, under 5 minutes and < 100 MB
     + Recommend you use OBS Studio (64bit) to record; make sure you test-watch your video!
     + See the Demo video posted in Canvas for an example of what’s expected here
  2. A **copy of this Word document**, with the following modifications:
     + Your submitted Perforce changelist # entered at the top of this section, at “My Perforce changelist # **???**” – this is the Perforce changelist # I should Get, run, test, and grade
     + Each line in the “Requirements” section with (X points) **must** be highlighted:
       - Fully completed requirements are highlighted cyan
         * for features you believe you’ve met/reproduced nearly exactly
       - Partially completed requirements are highlighted yellow
         * for features done but lacking or differing significantly vs. demo
       - Missing requirements are highlighted red
         * for features not implemented (not working or not attempted)