|  |
| --- |
| City, University of London |
| Asteroids Coursework |
| IN2026: Games Technology |

|  |
| --- |
| Jenson Bolton  4-5-2025 |

Table of Contents

[Coursework Objectives 2](#_Toc197232485)

[Implemented Features 2](#_Toc197232486)

[Overview 3](#_Toc197232487)

[Implemented Features 3](#_Toc197232488)

[Code Artefacts 3](#_Toc197232489)

[New Classes 3](#_Toc197232490)

[GamerStateManager 3](#_Toc197232491)

[New Methods in Asteroids 3](#_Toc197232492)

[Modified Methods in Asteroids 4](#_Toc197232493)

[New Member Variables in Asteroids 4](#_Toc197232494)

[Initial Failed Attempt 4](#_Toc197232495)

[Creating the Menu 4](#_Toc197232496)

[Stage I 4](#_Toc197232497)

[Adding Buttons 4](#_Toc197232498)

[Adding Buttons to the Window 5](#_Toc197232499)

[Working Attempt 5](#_Toc197232500)

[Setting up Game States 5](#_Toc197232501)

[Creating the Menu 6](#_Toc197232502)

[Creating the Instructions Menu 6](#_Toc197232503)

[Creating the High-Score Table 7](#_Toc197232504)

[Creating the Death Screen 7](#_Toc197232505)

[Black Holes 9](#_Toc197232506)

[Weapon Upgrade 9](#_Toc197232507)

[Extra Lives 9](#_Toc197232508)

Overview

# Coursework Objectives

# Implemented Features

Part I – Start Menu

# Overview

The Part I implementation introduces a modular, state-driven menu system into the existing Asteroids game without altering the core engine loop. At its heart lies a dedicated GameStateManager that tracks discrete modes - such as START\_MENU, PLAYING, SHOW\_INSTRUCTIONS, SHOWING\_HIGHSCORES and GAME\_OVER - and notifies registered listeners whenever the active mode changes. This approach cleanly separates user-interface concerns from gameplay logic, ensuring that menu overlays, instructional panels, high-score displays and the death-screen all coexist alongside the live asteroid simulation.

Upon launching the application, the game immediately enters the START\_MENU state. A backdrop of floating asteroids is spawned purely for visual ambience, and a full-screen GUIContainer is added to house the menu entries. Each option - Game Start, Difficulty toggle, Instructions, High-Scores and Quit - is represented by a centred GUILabel, with the current selection highlighted in yellow. Both keyboard and mouse events drive navigation: arrow keys and cursor hover update the selection index, while Enter, Space or a click activates the chosen item, causing a state transition.

Transitioning into PLAYING tears down the menu GUI and restores the classic Asteroids experience by spawning the player’s ship, displaying the score and lives tally, and adding game-world listeners for input and scoring. Later, when the player’s last life is lost, the GAME\_OVER state presents a tag-entry interface that collects a gamer tag (up to twelve characters), defaults to “Anon” if left blank, and appends the score to a persistent text file before displaying the SHOWING\_HIGHSCORES screen. Each menu state leverages the same underlying container and label infrastructure, ensuring consistency of appearance and behaviour throughout the game.

# Implemented Features

* A full screen start menu overlayed on floating asteroids, with neither the spaceship nor the score/lives HUD visible.
* A “Game Start” option that removes the menu, spawns the player’s spaceship, and reveals the score and lives labels.
* A “Difficulty” toggle that enables or disables bonuses/power-ups.
* A “Gameplay Instructions” entry that displays an overlay of control instructions with a Back button.
* A “High-Score Table” entry that reads from a text file, shows the top ten scores, and offers a Back button.
* Upon losing the final life, a “GAME OVER” screen prompts for a gamer tag (up to twelve characters: a–z, 0–9, underscore), defaults empty submissions to “Anon”, writes the record to the file, and then transitions into the high-score display.
* Full keyboard (arrow keys, Enter, Esc) navigation across all menu screens.

# Code Artefacts

The following tables catalogue all new and modified classes, methods and member variables introduced in Part I, together with their source files.

## New Classes

|  |  |  |
| --- | --- | --- |
| Class | Header | Implementation |
| *GameStateManager* | *GameStateManager.h* | *GamerStateManager.cpp* |

## GamerStateManager

|  |  |  |
| --- | --- | --- |
| Method | Declared In | Defined In |
| *GameStateManager()* | GameStateManager.h | GameStateManager.cpp |
| *~GameStateManager()* | *GameStateManager.h* | *GameStateManager.cpp* |
| *GetState() const* | *GameStateManager.h* | *GameStateManager.cpp* |
| *SetState(GameState)* | *GameStateManager.h* | *GameStateManager.cpp* |
| *AddListener(const Listener&)* | *GameStateManager.h* | *GameStateManager.cpp* |

## New Methods in Asteroids

|  |  |  |
| --- | --- | --- |
| Method | Declared In | Defined In |
| *ChangeState(GameState)* | *Asteroids.h* | *Asteroids.cpp* |
| *CreateStartMenu()* | *Asteroids.h* | *Asteroids.cpp* |
| *InitializeGameplay(shared\_ptr<Asteroids>)* | *Asteroids.h* | *Asteroids.cpp* |
| *CreateGameOverMenu()* | *Asteroids.h* | *Asteroids.cpp* |
| *ShowHighScoreTable()* | *Asteroids.h* | *Asteroids.cpp* |
| *ShowInstructions()* | *Asteroids.h* | *Asteroids.cpp* |
| *UpdateMenuHighlight()* | *Asteroids.h* | *Asteroids.cpp* |

## Modified Methods in Asteroids

|  |  |
| --- | --- |
| Method | Source File |
| *Start()* | *Asteroids.cpp* |
| *Stop()* | *Asteroids.cpp* |
| *OnKeyPressed(uchar, int, int)* | *Asteroids.cpp* |
| *OnKeyReleased(uchar, int, int)* | *Asteroids.cpp* |
| *OnSpecialKeyPressed(int, int, int)* | *Asteroids.cpp* |
| *OnSpecialKeyReleased(int, int, int)* | *Asteroids.cpp* |
| *OnScoreChanged(int)* | *Asteroids.cpp* |
| *OnPlayerKilled(int)* | *Asteroids.cpp* |
| *CreateAsteroids(const uint)* | *Asteroids.cpp* |
| *CreateGUI()* | *Asteroids.cpp* |

## New Member Variables in Asteroids

|  |  |  |
| --- | --- | --- |
| Member | Type | Declared In |
| *mStateManager* | *GameStateManager* | *Asteroids.h* |
| *mMenuContainer* | *shared\_ptr<GUIContainer>* | *Asteroids.h* |
| *mMenuLabels* | *vector<shared\_ptr<GUILabel>>* | *Asteroids.h* |
| *mMenuSelection* | *int* | *Asteroids.h* |
| *mBonusesEnabled* | *bool* | *Asteroids.h* |
| *mMenuAsteroids* | *vector<shared\_ptr<GameObject>>* | *Asteroids.h* |
| *mCurrentTag* | *string* | *Asteroids.h* |
| *mTagLabel* | *shared\_ptr<GUILabel>* | *Asteroids.h* |
| *MAX\_TAG\_LEN* | *static const size\_t* | *Asteroids.h* |

# Initial Failed Attempt

## Creating the Menu

### Stage I

To create the menu screen, I have created a StartMenu class based on the Asteroids class. The reasoning behind this is so that I can keep it separate from the rest of the game and prevent any UI errors. This may be integrated into a scene manager system that will have the start screen, difficulty menu, main game, and death screen. Currently, this is made separate to build out the menu properly before providing the buttons with their intended functions.

## Adding Buttons

To create clickable buttons, a GUIButton class was defined. It maintains:

* A std::string for the button label.
* A std::function<void()> callback for clicking actions.
* A Boolean hover flag to toggle visual state.

Its constructor centres its label, assigns a default size 200x40 pixels (this has been adopted as standard size, dynamic sizing is not necessary for this project’s size), and initialises the hover state. The Draw() method renders a background rectangle and border using OpenGL primitives, before drawing the label with GLUT’s bitmap font. The OnMousePressed(int mx, int my, int button) tests for left button clicks within its bounds, invoking the stored callback when activated.

### Adding Buttons to the Window

Within the StartMenu class, the CreateGUI() method preforms the following steps to create a start menu:

* Retrieves the engine’s GUIContainer via mGameDisplay->GetContainer() and sets an internal border.
* Iterates over a list of menu entries, creating a GUIButton for each, assigning their respective callbacks (currently place holders except quit, which will call this->Stop()).
* Computes normalised positions so that each button is centred at 50% of the width and spaced vertically at 15% intervals.

# Working Attempt

## Setting up Game States

For this version of the start menu, I introduced a dedicated GameStateManager class which now encapsulates all valid states - namely START\_MENU, PLAYING, GAME\_OVER, SHOWING\_HIGHSCORES and SHOW\_INSTRUCTIONS - and provides listener registration for state transitions. This new class resides in GameStateManager.h/.cpp and isolates state logic from the game session itself.

Originally, I tried to achieve this by defining the enum in Asteroids class but instead decided to separate this into its own class so that the listeners are coded in a clear manner. Now Asteroids includes GameStateManager.h and declares helper methods for each state: CreateStartMenu(), InitializeGameplay(), CreateGameOverMenu(), ShowHighScoreTable() and ShowInstructions(). Internally, a GameStateManager mStateManager; member was added, together within wrappers RegisterStateListener() and UnregisterStateListener() to forward listeners into the manager. A private ChangeState(GameState) method was introduced to trigger transitions.

Asteroids.cpp underwent the most substantial changes. The original Start() function was refactored to register a single listener lambda that dispatches on the current state via a switch statement:

RegisterStateListener([thisPtr](GameState state) {

switch (state) {

case GameState::START\_MENU:

thisPtr->CreateStartMenu();

break;

case GameState::PLAYING:

thisPtr->InitializeGameplay(thisPtr);

break;

case GameState::GAME\_OVER:

thisPtr->CreateGameOverMenu();

break;

case GameState::SHOWING\_HIGHSCORES:

thisPtr->ShowHighScoreTable();

break;

case GameState::SHOW\_INSTRUCTIONS:

thisPtr->ShowInstructions();

break;

default:

break;

}

});

ChangeState(GameState::START\_MENU);

With this dispatch in place, each stubbed method can focus on its own UI concerns. For example, CreateStartMenu() will later display the floating-asteroid backdrop and menu buttons, InitializeGameplay() performs the original game setup (spaceship and asteroids spawn, GUI labels, score/listener registrations), and ShowInstructions() will overlay control instructions with an option to return to the start menu. To verify the state machine, the current implementation of CreateStartMenu() immediately calls ChangeState(GameState::PLAYING) and invokes the original setup logic, ensuring that the transition mechanism functions correctly before any further menu elements are added.

## Creating the Menu

The start menu is constructed as an overlay on the existing game world without altering the core engine. A handful of asteroids is spawned purely for visual effect, drifting across the backdrop in the START\_MENU state. A full‐screen GUIContainer is then created and added to the root display, with its size matched to the game window and a uniform border to frame the menu. Within this container, five GUILabel widgets are laid out at evenly spaced vertical positions, each centred horizontally. These labels read “Game Start”, “Bonuses: On” (or “Off” as toggled), “Instructions”, “High-Scores” and “Quit”. The first entry is immediately highlighted in yellow to indicate initial focus.

Navigation through the menu is handled by keyboard listeners. Pressing the up or down arrows adjusts an integer index, wrapping explicitly between the first and last entries to avoid any risk of negative underflow, and calls UpdateMenuHighlight() to recolour the labels accordingly. Hitting Enter or Space on the selected label, invokes ActivateMenuItem(), which either changes the game state (to PLAYING, SHOW\_INSTRUCTIONS or SHOWING\_HIGHSCORES), toggles the bonus flag and updates the second label’s text, or calls Stop() when “Quit” is chosen.

When “Game Start” is activated, the menu container is removed from the display and its shared pointer reset, clearing all menu widgets before gameplay begins. InitializeGameplay() then performs the original setup: registering the score keeper, spawning the player’s spaceship, creating the initial asteroids, and drawing the score and lives labels. This sequence ensures a seamless transition from the handcrafted menu overlay into the familiar Asteroids experience without ever restarting the underlying session loop.

### Code Snippet

void Asteroids::CreateStartMenu() {

// Remove any other menus

if (mMenuContainer) {

mGameDisplay->GetContainer()->RemoveComponent(mMenuContainer);

}

else {

CreateAsteroids(8);

}

// Add asteroids if there are none

// State occurs after game over

if (asteroids.empty()) {

CreateAsteroids(40);

}

mMenuContainer = make\_shared<GUIContainer>();

mMenuContainer->SetSize(mGameDisplay->GetContainer()->GetSize());

mMenuContainer->SetBorder({ 50,50 });

mGameDisplay->GetContainer()->AddComponent(mMenuContainer, { 0.0f, 0.0f });

vector<string> entries = {

"Game Start",

string("Bonuses: ") + (mBonusesEnabled ? "On" : "Off"),

"Instructions",

"High-Scores",

"Quit"

};

mMenuLabels.clear();

for (int i = 0; i < (int)entries.size(); ++i) {

auto lbl = make\_shared<GUILabel>(entries[i]);

lbl->SetHorizontalAlignment(GUIComponent::GUI\_HALIGN\_CENTER);

lbl->SetVerticalAlignment(GUIComponent::GUI\_VALIGN\_MIDDLE);

// relative positions: x = 0.5 (centre); y = 0.7, 0.5, 0.3, 0.1

float y = 0.7f - i \* 0.1f;

mMenuContainer->AddComponent(lbl, { 0.5f, y });

mMenuLabels.push\_back(lbl);

}

mMenuSelection = 0;

UpdateMenuHighlight();

}

## Creating the Instructions Menu

The instructions overlay preserves the floating‐asteroid backdrop while replacing the start‐menu widgets with a focused set of guidance labels and a single navigable “Back” option. On entering the SHOW\_INSTRUCTIONS state, any existing container is detached and its shared pointer reset, ensuring a clean slate for the new GUI components. A fresh full-screen GUIContainer is then instantiated with a consistent border, matching the game display dimensions. Five individual GUILabel instances are created - each conveying a single line of control guidance (“<- / -> to turn”, “^ to thrust”, “Space to shoot”, “Click or press Enter to select”, and “Press Esc to quit”) - and are centred horizontally, spaced evenly down the upper portion of the container. This approach avoids relying on unsupported newline sequences.

Beneath the instruction lines, a distinct “Back” label is added at the base of the container and registered as the sole entry in the mMenuLabels vector. Keyboard and mouse listeners detect Enter, Space or a left‐click on this label to invoke a state transition back to START\_MENU, while pressing Esc triggers an immediate call to Stop(), terminating the session. Selection highlighting is managed through the existing UpdateMenuHighlight() routine, ensuring the “Back” option is always visibly distinguished. By isolating the instructions UI in its own state handler, the solution maintains the integrity of the core game loop and sets the stage for seamless navigation between menus and gameplay.

### Code Snippet

void Asteroids::ShowInstructions() {

// Remove the main menu but keep the asteroids

mGameDisplay->GetContainer()->RemoveComponent(mMenuContainer);

// Full screen container

mMenuContainer = make\_shared<GUIContainer>();

mMenuContainer->SetSize(mGameDisplay->GetContainer()->GetSize());

mMenuContainer->SetBorder({ 50, 50 });

mGameDisplay->GetContainer()->AddComponent(mMenuContainer, { 0.0f, 0.0f });

// Instructions

vector<string> lines = {

"<-/-> to turn",

"^ to thrust",

"Space to shoot",

"Click or press Enter to select",

"(Press Esc to quit)"

};

float startY = 0.65f;

float deltaY = 0.06f;

for (int i = 0; i < (int)lines.size(); ++i) {

auto lbl = make\_shared<GUILabel>(lines[i]);

lbl->SetHorizontalAlignment(GUIComponent::GUI\_HALIGN\_CENTER);

lbl->SetVerticalAlignment(GUIComponent::GUI\_VALIGN\_MIDDLE);

float y = startY - i \* deltaY;

mMenuContainer->AddComponent(lbl, { 0.5f, y });

}

// Back button

mMenuLabels.clear();

auto backLbl = std::make\_shared<GUILabel>("Back");

backLbl->SetHorizontalAlignment(GUIComponent::GUI\_HALIGN\_CENTER);

backLbl->SetVerticalAlignment(GUIComponent::GUI\_VALIGN\_MIDDLE);

mMenuContainer->AddComponent(backLbl, { 0.5f, 0.2f });

mMenuLabels = vector<shared\_ptr<GUILabel>>{ backLbl };

// Selection and higlighting

mMenuSelection = 0;

UpdateMenuHighlight();

}

## Creating the High-Score Table

The high-score screen begins by detaching any existing GUI components from the display, ensuring that the instructions or main menu overlays are fully removed. A fresh, full-screen GUIContainer is then instantiated with a consistent border matching the game window dimensions. At the top of this container, a centred “High-Scores” header is added to clearly denote the purpose of the view.

Persistent data is sourced from a simple text file named highscores.txt, in which each line comprises a gamer tag and its associated numeric score. Upon entering the high-score state, the file is opened, and every tag-score pair is read into memory. These entries are sorted in descending order by score and truncated to the top ten results.

Each of the ten rows is rendered as a centred GUILabel, aligned vertically down the container, with text in the form “n. TAG – SCORE”. A single navigable “Back” label appears at the base of the screen and is the sole member of the menu-labels vector. Keyboard listeners detect Enter, Space on this label; any of these interactions triggers a state change back to the start menu. This design preserves the integrity of the game loop, cleanly separates persistent data handling from rendering logic, and offers a straightforward mechanism for players to review and return without restarting the session.

### Code Snippet

void Asteroids::ShowHighScoreTable() {

// clear old GUI

if (mMenuContainer) {

mGameDisplay->GetContainer()->RemoveComponent(mMenuContainer);

mMenuContainer.reset();

mMenuLabels.clear();

}

// container framing

mMenuContainer = make\_shared<GUIContainer>();

mMenuContainer->SetSize(mGameDisplay->GetContainer()->GetSize());

mMenuContainer->SetBorder({ 50,50 });

mGameDisplay->GetContainer()->AddComponent(mMenuContainer, { 0.0f,0.0f });

// header

auto header = make\_shared<GUILabel>("High-Scores");

header->SetHorizontalAlignment(GUIComponent::GUI\_HALIGN\_CENTER);

header->SetVerticalAlignment(GUIComponent::GUI\_VALIGN\_TOP);

mMenuContainer->AddComponent(header, { 0.5f, 0.9f });

// load and display the entries

auto scores = LoadHighScores();

for (int i = 0; i < 10; ++i) {

string text;

if (i < (int)scores.size()) {

text = to\_string(i + 1) + ". "

+ scores[i].first + " – "

+ to\_string(scores[i].second);

}

else {

text = to\_string(i + 1) + ". ——";

}

auto lbl = make\_shared<GUILabel>(text);

lbl->SetHorizontalAlignment(GUIComponent::GUI\_HALIGN\_CENTER);

lbl->SetVerticalAlignment(GUIComponent::GUI\_VALIGN\_MIDDLE);

float y = 0.8f - i \* 0.07f;

mMenuContainer->AddComponent(lbl, { 0.5f, y });

}

// “Back” button

auto backLbl = std::make\_shared<GUILabel>("Main Menu");

backLbl->SetHorizontalAlignment(GUIComponent::GUI\_HALIGN\_CENTER);

backLbl->SetVerticalAlignment(GUIComponent::GUI\_VALIGN\_MIDDLE);

mMenuContainer->AddComponent(backLbl, { 0.5f, 0.05f });

// only this label is navigable

mMenuLabels = { backLbl };

mMenuSelection = 0;

UpdateMenuHighlight();

}

## Creating the Game Over Screen

Upon the player’s final life being lost, the GAME\_OVER state replaces all prior GUI overlays with a focused tag‐entry interface. Any existing container is removed from the display and its pointer reset, ensuring that no remnants of the start or instructions menus remain. A new full-screen GUIContainer is instantiated with a consistent fifty‐pixel border and matched to the window dimensions. Centre‐aligned at the top of this pane is a prominent “GAME OVER” label, immediately followed by a prompt reading “Enter your tag:” placed slightly lower to guide the player into the input field.

Directly beneath the prompt sits an initially blank, centre‐aligned GUILabel dedicated to reflecting the player’s current tag entry in real time. This widget is held in its own member, mTagLabel, so that every valid keystroke—letters a–z, digits 0–9 and underscore up to a maximum of twelve characters—immediately updates the visible text. Backspace removes the last character, and any unsupported keys are ignored. The editable label prevents accidental overwriting of navigation elements and isolates the input logic for clarity.

A “Submit” label appears below the entry field and is the sole navigable element in this state. Selection highlighting is managed via the existing mMenuLabels vector and UpdateMenuHighlight(), ensuring that the “Submit” button remains visually distinct. Pressing Enter when it is highlighted writes the tag and current score to the persistent text file. An empty submission defaults to “Anon” before saving. Once written, the state machine transitions seamlessly to SHOWING\_HIGHSCORES without restarting the session loop, thus maintaining continuity of the game’s world update sequence.

### Code Snippet

void Asteroids::CreateGameOverMenu() {

// clear old GUI

if (mMenuContainer) {

mGameDisplay->GetContainer()->RemoveComponent(mMenuContainer);

mMenuContainer.reset();

mMenuLabels.clear();

}

for (auto& a : asteroids) {

mGameWorld->RemoveObject(a);

}

asteroids.clear(); // remove menu asteroids

// container framing

mMenuContainer = make\_shared<GUIContainer>();

mMenuContainer->SetSize(mGameDisplay->GetContainer()->GetSize());

mMenuContainer->SetBorder({ 50,50 });

mGameDisplay->GetContainer()->AddComponent(mMenuContainer, { 0.0f,0.0f });

// Heading

auto overLbl = make\_shared<GUILabel>("GAME OVER");

overLbl->SetHorizontalAlignment(GUIComponent::GUI\_HALIGN\_CENTER);

overLbl->SetVerticalAlignment(GUIComponent::GUI\_VALIGN\_TOP);

mMenuContainer->AddComponent(overLbl, { 0.5f, 0.85f });

// Prompt

auto prompt = make\_shared<GUILabel>("Enter your tag:");

prompt->SetHorizontalAlignment(GUIComponent::GUI\_HALIGN\_CENTER);

prompt->SetVerticalAlignment(GUIComponent::GUI\_VALIGN\_MIDDLE);

mMenuContainer->AddComponent(prompt, { 0.5f, 0.65f });

// editable tag

mCurrentTag.clear();

auto tagLbl = make\_shared<GUILabel>("");

tagLbl->SetHorizontalAlignment(GUIComponent::GUI\_HALIGN\_CENTER);

tagLbl->SetVerticalAlignment(GUIComponent::GUI\_VALIGN\_MIDDLE);

mMenuContainer->AddComponent(tagLbl, { 0.5f, 0.55f });

mTagLabel = tagLbl;

// Submit btn

auto submitLbl = make\_shared<GUILabel>("Submit");

submitLbl->SetHorizontalAlignment(GUIComponent::GUI\_HALIGN\_CENTER);

submitLbl->SetVerticalAlignment(GUIComponent::GUI\_VALIGN\_MIDDLE);

mMenuContainer->AddComponent(submitLbl, { 0.5f, 0.35f });

// Submit navigation

mMenuLabels = { submitLbl };

mMenuSelection = 0;

UpdateMenuHighlight();

mMenuLabels.push\_back(tagLbl);

}

# UI Screen Shots

## Main Menu

A screenshot of a computer

AI-generated content may be incorrect.

## Instructions Menu

A screenshot of a computer

AI-generated content may be incorrect.

## High Score Table

A screenshot of a computer

AI-generated content may be incorrect.

## Game Over

A screenshot of a computer

AI-generated content may be incorrect.

## Main Menu After Death

A screenshot of a computer screen

AI-generated content may be incorrect.

Part II

# Black Holes

## Overview

The Black Hole feature introduces a temporary, randomly spawning gravity well into the world that:

* Spawns at uniformly distributed positions within the game bounds, every 10-20 s.
* Attracts both asteroids and the player’s spaceship toward its centre with strength ∝ 1/(distance²).
* Persists for a configurable duration before self-destroying.
* Displays a dedicated blackhole.png sprite loaded via the AnimationManager.

## Implemented Features

* Random timer-driven spawns: uses a new SPAWN\_BLACKHOLE timer (ID 1001), seeded once in Start().
* Configurable parameters: radius, strength and duration are constants in OnTimer().
* Collision-based pull: the hole’s bounding sphere triggers OnCollision each frame, applying acceleration to overlapping Asteroid and Spaceship objects.
* Sprite rendering: the hole is rendered with a static looping animation and initialised by calling Reset() immediately after SetSprite().

## Code Artefacts

### New Classes

|  |  |  |  |
| --- | --- | --- | --- |
| Class | Header | Implementation | Purpose |
| *Bonus* | *Bonus.h* | *Bonus.cpp* | *Base for pick-ups; lifetime, removal on collision.* |
| *BlackHoleBonus* | *BlackHoleBonus.h* | *BlackHoleBonus.cpp* | *Spawns a BlackHole on ship contact; then self-removes.* |
| *BlackHole* | *BlackHole.h* | *BlackHole.cpp* | *Gravity-well effect; lifetime, collision-test, pull logic.* |

### New Methods in Asteroids

|  |  |  |  |
| --- | --- | --- | --- |
| Method / Constant | Declared In | Defined In | Description |
| *static const int SPAWN\_BLACKHOLE = 1001;* | *Asteroids.cpp* | *same* | *Timer ID for random Black-Hole spawns.* |
| *void Asteroids::Start() (mod)* | *Asteroids.cpp* | *same* | *Calls srand(time(nullptr)), loads blackhole animation, schedules first SPAWN\_BLACKHOLE.* |
| *void Asteroids::OnTimer(int) (ext)* | *Asteroids.cpp* | *same* | *Handles SPAWN\_BLACKHOLE: picks random (x,y), creates/configures BlackHole, re-schedules itself.* |

### Modified Methods in Asteroids

|  |  |  |
| --- | --- | --- |
| Method | Source File | Change |
| *Asteroids::OnObjectRemoved* | *Asteroids.cpp* | ***Removed*** *old asteroid-death bonus block so holes only spawn via timer.* |
| *Asteroids::Start()* | *Asteroids.cpp* | ***Added*** *srand(...), AnimationManager::CreateAnimationFromFile("blackhole",32,32,32,32,"blackhole.png"), timer.* |
| *Asteroids::OnTimer(int)* | *Asteroids.cpp* | ***Extended*** *to catch SPAWN\_BLACKHOLE, randomise position, spawn/configure hole (shape, sprite, Reset(), add to world).* |

### Code Snippet

else if (value == SPAWN\_BLACKHOLE && mBonusesEnabled) {

float hw = mGameWorld->GetWidth() \* 0.5f;

float hh = mGameWorld->GetHeight() \* 0.5f;

float x = (rand() / float(RAND\_MAX)) \* 2 \* hw - hw;

float y = (rand() / float(RAND\_MAX)) \* 2 \* hh - hh;

const float holeRadius = 50.0f;

const float holeStrength = 400.0f;

const int holeDuration = 5000;

auto hole = std::make\_shared<BlackHole>(

GLVector3f{ x,y,0.0f },

holeRadius,

holeStrength,

holeDuration

);

hole->SetBoundingShape(

std::make\_shared<BoundingSphere>(hole->GetThisPtr(), holeRadius)

);

Animation\* bhAnim = AnimationManager::GetInstance()

.GetAnimationByName("blackhole");

auto bhSprite = std::make\_shared<Sprite>(

bhAnim->GetWidth(), bhAnim->GetHeight(), bhAnim

);

bhSprite->SetLoopAnimation(true);

hole->SetSprite(bhSprite);

hole->SetScale(0.5f);

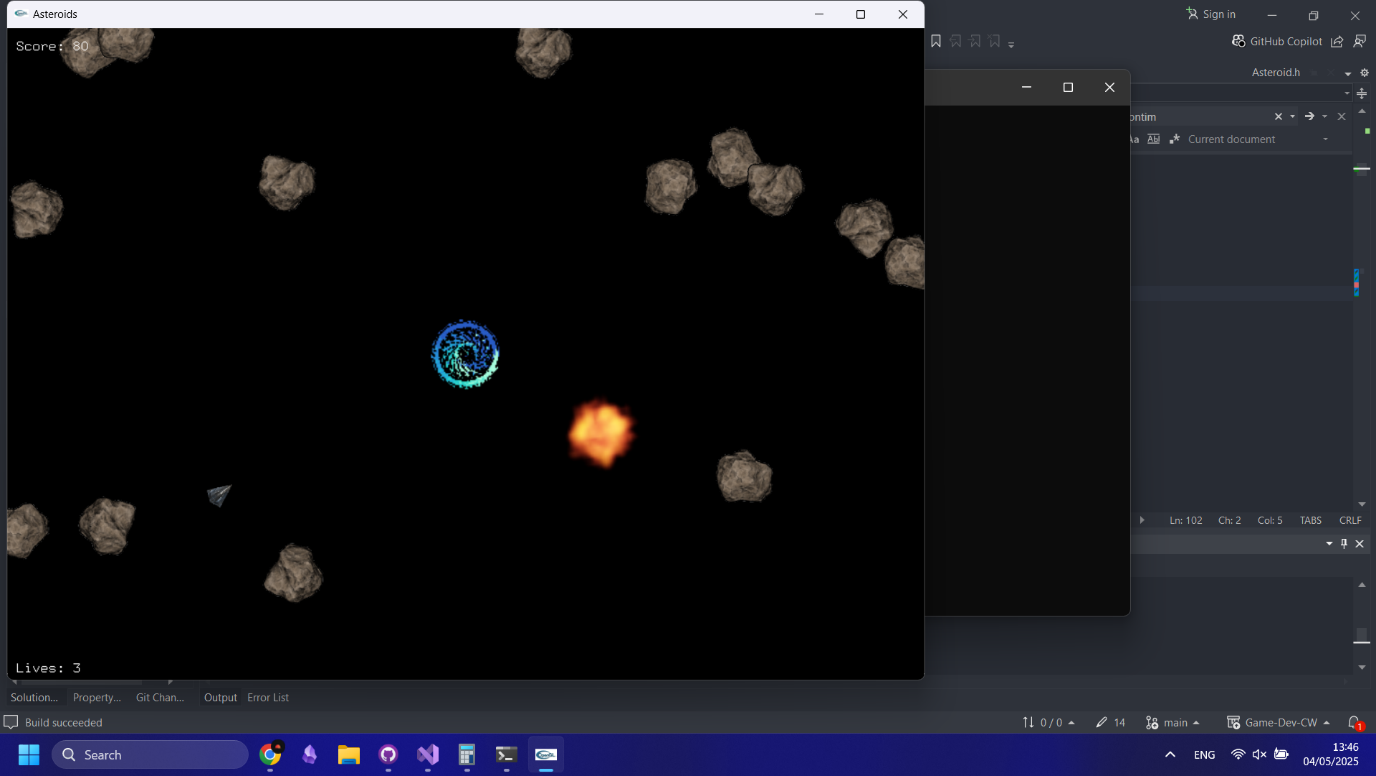
hole->Reset();

mGameWorld->AddObject(hole);

SetTimer(10000 + (rand() % 10001), SPAWN\_BLACKHOLE);

}

## Screenshot



# Weapon Upgrade

## Overview

The Weapon Upgrade bonus grants the player a temporary "dual-shot" ability, causing the spaceship to fire bullets both forward and backward for a fixed duration when collected.

## Implemented Features

* Pickup spawning: if bonuses are enabled, a white square-shaped bonus spawns at the location of a destroyed asteroid (10% chance).
* Collision: the bonus uses a BoundingSphere (radius 5 units) to detect collision with the spaceship.
* Activation: on collision, the bonus removes itself and calls Spaceship::ActivateDualShot(duration\_ms) to grant dual shot for 10 seconds.
* Dual - shot logic: while active, Spaceship::Shoot() spawns an additional bullet opposite the ship’s facing, using the same speed.

## Code Artefacts

### New Class: WeaponUpgradeBonus

|  |  |  |  |
| --- | --- | --- | --- |
| Class | Header | Source | Purpose |
| *WeaponUpgradeBonus* | *WeaponUpgradeBonus.h* | *WeaponUpgradeBonus.cpp* | *Subclass of Bonus; on collision, activates dual-shot and self-removes.* |

### Spaceship Modifications

# Extra Lives