**OOP – Final Project C#**

Ido Drutin

**The game is a Top Down Survivor IO – shooting works like Survivor IO, but it doesn’t get harder as time goes on.**

Movement is **WASD** based, there is no other button in the game besides **escape** to exit the game.

Your as a box, there are Triangles, Polygons and circles coming to kill you, if they manage to get to you they will kill you quite rapidly.

You need to navigate around the randomly generated gray boxes (Static Objects) and randomly generated chests. you need to shoot through teal chests to receive a buff, currently there are to buffs extra bullets up to 3 in total and movement speed that has no cap currently.

survive for as long as you can to get the highest score (kill count) possible.

The main game mechanic is movement, so you can survive as long as possible.

**Opening The Game**

To open the game all you need to do is open visual studio, and run the game, there are no other libraries involved besides the regular monogame framework we downloaded in the course.

**Main Classes**

* **Game1**
  + Makes background tiling, enemy spawning, collision management, and UI rendering.

**Player and Characters**

* **Character (Abstract):**
  + Base class for all characters (player and enemies) that has properties (position, speed, health) and methods (drawing, collision checking, taking damage).
* **Player**
  + Inherits from Character and adds IShootable.
  + Handles player movement, collision with static objects, shooting (with extra bullet buff mechanics), and applying buffs.

**Enemies**

* **BaseEnemy**
  + Inherits from Character.
  + Gives the ability to move toward the player while avoiding collisions with static objects, and random spawning.
* **TriangleEnemy, PolygonEnemy, CircleEnemy**
  + Classes from BaseEnemy with certain properties (health and speed) controlled by constants.
* **EnemySpawner**
  + Creating new enemies at random timings.
  + Uses constants for enemy health and speed, and randomly chooses the enemy type based on controlled randomness.

**Static and Breakable Objects**

* **StaticSpawn**
  + Implements IStatic for static obstacles in the game.
* **BreakableStatic**
  + Implements both IStatic and IBreakable.
  + Shows breakable objects that take damage and, when destroyed, generate a random buff for the player.

**Managers**

* **CollisionManager**
  + Handles collisions between the player and enemies, bullets and enemies, and triggers particle effects on enemy destruction.
* **GameManager**
  + Manages overall game state (Playing, Paused (didn’t manage to implement it in time), Game Over) and adds methods to reset or end the game.
* **ScoreManager**
  + Manages the players score, including methods to add points and reset the score.
* **HighScoreManager**
  + Implements file I/O for saving and loading the high score – Saves a Text File.
* **InputManager**
  + Methods to get player input (movement and shooting).

**Particles**

* **Particle**
  + Shows a simple particle with properties such as position, velocity, colour, and lifetime.
  + Has an update method to control movement and removal, and a draw method that uses a scale to enlarge the particle.
* **ParticleSystem**
  + Manages a collection of particles, updating and drawing them, and removing expired particles.

**Interfaces**

* **IDamageable**
  + Declares methods for taking damage and checking if an object is dead.
* **IShootable**
  + Declares the shooting method for objects that can shoot (used by the player).
* **IStatic**
  + Declares drawing and collision bounds methods for static objects.
* **IBreakable**
  + Declares methods for applying damage, checking destruction, and generating a buff when destroyed.