Instructions

Welcome to the take-home exercise portion of the Unity/C# interview process at Outplay Entertainment. These 3 exercises are designed to evaluate your problem-solving skills, code quality, and familiarity with Unity and C# development.

Unity Version

- Please use **Unity version 2022.3.30f1** for any tasks involving Unity. This ensures compatibility and consistency with our internal environment.

Version Control

- Create a GitHub or GitLab repository to host your solutions. We value incremental progress, so please commit your code frequently as you work through each task. Avoid pushing all changes in a single commit; this helps us understand your development process and thought patterns.
- Ensure that all your work is done directly in the repository. Proper use of version control is an important aspect of this exercise, and we encourage you to structure your project well and document your approach.

Original Work

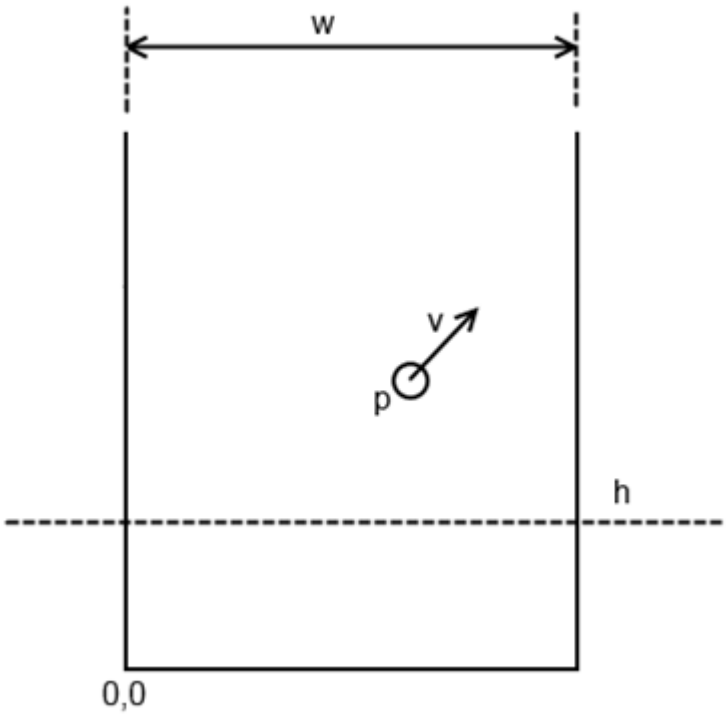
- We are looking for solutions that showcase your understanding of C# and Unity development. While there are many resources available online, we encourage you to approach these exercises as if you were solving them in a real-world scenario.
- During our review, we'll be interested in understanding your problem-solving process, so please ensure that your code reflects your personal approach and thinking.

Thank you for taking the time to complete these exercises. Once you have completed the test, please share the repository containing the responses in the answering to the email containing this document.

We look forward to reviewing your solutions.

Exercise 1

Imagine a 2D game where the playing area is an enclosed rectangle with width w and infinite height. In the playing area there is a ball. The ball is subject to a gravitational acceleration ***G***. The ball starts at position ***p*** and has an initial velocity of ***v***.



Write an implementation for a reusable function with the signature below which predicts the position the ball will be at if and when it reaches the specified height ***h***.

```
bool TryCalculateXPositionAtHeight(
       float h,
       Vector2 p,
       Vector2 v,
       float G,
       float w,
       ref float xPosition)
```

Assumptions:

- The playing area doesn't have any boundary at the top, so no collision ever happens with it.
- The ball can be considered as a point.
- There are no precision issues, and everything can easily fit in a float.
- There is no air resistance.
- All collisions are perfect without any energy loss.
- If the ball hits the bottom it won't bounce.

Exercise 2

You are working on a match-3 game with the following rules:

- Pairs of jewels adjacent vertically and horizontally can be swapped.
- You can only swap jewels when this will result in a match being created.
- A match happens when there are 3 or more jewels of the same kind adjacent vertically or horizontally.
- All jewels involved in matches are set to *JewelKind::Empty* after each move.
- One point is given for each jewel that has been removed. The best move for a given board is thus the one that will remove the most jewels.
- The initial board state contains no matches; therefore, swapping jewels is the only way matches can be created.

Given the code below implement the *CalculateBestMoveForBoard* function.

```
public class Board
{
    enum JewelKind
    {
        Empty,
        Red,
        Orange,
        Yellow,
        Green,
        Blue,
        Indigo,
        Violet
    }

    enum MoveDirection
    {
        Up,
        Down,
        Left,
        Right
    }

    struct Move
    {
        public int x;
        public int y;
        public MoveDirection direction;
    }

    int GetWidth();
    int GetHeight();
    JewelKind GetJewel(int x, int y);
    void SetJewel(int x, int y, JewelKind kind);


    Move CalculateBestMoveForBoard()
    {
        // Implement this function
    }
}
```

Exercise 3

Create a simple Unity project using C# with the following requirements:

- In the scene there are 100 objects that are spawned at random positions that can collide with a main game object.
- The main game object moves at a constant speed, starting at (0, 0, 0), towards 3 points (P1, P2, P3) that can be changed by a designer.
- At arrival at each point, the game object should begin movement toward the next point.
- On arrival at the final point or if a collision is detected, the main game object should be removed from the scene, play a sound effect and show a particle effect.
- Note that the game uses the physics system provided by Unity and the collisions are reported by Unity