SUTD GAME LAB

Game Innovation Programme
Game Programmer Test

Name: ________________________________

School: ________________________________

Instructions

Thank you for applying for the Game Programmer position. Please do not freak out over the word 'test'; it is more like an assessment tool for us to have a better understanding of your competency and ability.

Please read the questions and instructions carefully and as with any other tests/exams, double-check before submission. If you have any questions, you may contact us at **gamelab@sutd.edu.sg**

Create a README text file. Kindly put in your full name and school. List down any extra credit features attempted as well as what the new gameplay feature is from Task #3. Explain why your feature makes the game a whole lot more fun. Zip up the README text file and project folder with all of your changes. Name the zipped file **GIP2015_Programmer_YourFullName.zip**. Upload the zip file to a file transfer service (e.g. Dropbox) and submit the link to the web page indicated in the confirmation email.

*Note: Since this is a take-home test, you're encouraged to research online or consult others. However, we will not accept answers that are obviously plagiarized.

Section ONE: Important information

1. Short Introduction

The game programmer intern will be fully involved in the game development process, as well as owning the technical aspects of the game project. As we will be diving straight into development right from the start of the programme, we will expect the selected candidate to be able to pick up concepts easily with little guidance.

Unity3D will be the main platform used to create games. This test is designed to

- assess the candidates’ game programming ability
- assess the candidates’ ability to learn Unity on his/her own
- familiarise candidates with Unity’s workflow

During the project, you will be working with other programmers as part of a team. The other programmers on your team need to understand the code you have written, and you need to understand the code that other programmers have written. Often, we continue these projects into the rest of the year, and this means a whole new set of programmers will come in and try to use your code. Many of these programmers will not be familiar with the language we are using.

All of this means that other humans should be able to read your code, and they should be able to do so quickly and effortlessly. Those other humans should not have to get into the details of your code unless they are hunting bugs.

2. Tips on Readable Code

Programmers can argue endlessly about the Right Way to write readable code, but at our Game Lab, we'll start with a few basic rules:

- Write comments to describe WHY we are doing something.
- Write comments to describe WHAT we are doing only when the briefest glance at the code is not enough.
- Name variables and functions so that they remove the need for comments. This usually requires long, descriptive names.
- If an algorithm is long enough that it's easy to lose track of the steps, write an overview of the algorithm in plain language at the top.
- If you aren't sure whether or not something needs a comment, use a comment.

Some examples:

//// (A)

```
if (x < 0 || x >= width)
```

```
          return null;
```

//// (B)

```
// Is x out of bounds?
if (x < 0 || x >= width)
  // Yes!
     return null;
```

(A) is decent code. The WHAT is pretty clear.

(B) makes the WHY so clear that we don't need to read the conditional expression at all. Better!

//// (C)

```
if ((v-v2).getLength() < 30)
     return 50;
```

//// (D)

```
// Is the distance between arrow and target small enough?
if ((v-v2).getLength() < 30)
     return 50;  // bulls eye!
```

//// (E)

```
Vector2 offset_from_center = arrow_location - center_of_target;
// Did the player get a bulls eye?
if (offset_from_center.getLength() < bulls_eye_radius)
     // Yes!
     return bulls_eye_score;
```

(C) You can read it and you can figure out what it is doing, but it's a bit cryptic. And we know nothing about WHY.

(D) Comments remove the need to read the code. We learn something about the surrounding code just by glancing at this one snippet: we are dealing with someone shooting at a target, and this piece handles bulls eyes. Better!

(E) Covers the high level WHY in comments. If you want to know WHAT it is doing, the code documents itself through descriptive variable names and constants. Not only do we know that we are handling bulls eyes, we also know that the surrounding code is probably a scoring routine, we can guess that other possible scores are held in variables ending in _score, and that other possible rings in the target might be described by variables ending in _radius. Better still!

In writing **(E),** we talked about using the temporary "offset_from_center" variable. Was the snippet really easier to read that way? In the end, we left "offset_from_center" in because it demonstrates a useful technique: a well-named temp variable can sometimes make code more readable. We could have gone either way.

3. Download Unity 4.6 [Free version] from
http://unity3d.com/unity/download/

Install the application.
Extract the file InternshipTestPlatformer.zip into a working project folder.

Run Unity 4.6.
Go to File > Open Project.
Locate the base project files downloaded and open the project.

For more information on how to use Unity 4.6, visit
http://docs.unity3d.com/Manual/index.html

Useful Script References
http://docs.unity3d.com/Documentation/ScriptReference/MonoBehaviour.html
http://docs.unity3d.com/Documentation/ScriptReference/GameObject.html
http://docs.unity3d.com/Documentation/ScriptReference/Transform.html
http://docs.unity3d.com/Documentation/ScriptReference/Rigidbody.html
Do look up more of the classes from this reference.

4. Gameplay Components

- ComponentHealth: Health (HP) before game object dies
- ComponentMovement: Keyboard control to move the game object
- ComponentShoot: Keyboard control to shoot. Instantiates a Bullet object and propels it in the direction the game object is facing with a predetermined speed.
- ComponentBullet: When colliding with player/enemy, deducts their health. Destroy self on collision
- ComponentMovingPlatform: Makes the platform moves by itself.
- ComponentCooldown: Calls a function whenever the cooldown is reached

UI

- UIHealthBar: Visuals for health bar on top of a game object. *Requires ComponentHealth.

Section TWO: Tasks

Task #1

This section of the code test is primarily intended to test communication and documentation skills, code comprehension, and the ability to follow instructions.

The comments and variable names in the script files have been deliberately changed to make the code difficult to comprehend. Go through these files and make them more readable again.

Task #2

Here is a list of features to implement into the base project. All scripts must be in C#.

Core Features

- Moving left and right
  - At the moment, our player is a sitting duck. Implement the movement to move our player left and right. The player should be moved by holding down the left and right arrow keys.
- Jumping
  - Moving horizontally alone isn't enough; our player need to jump. If not, it won't be called a platformer test. The player needs to jump when the up arrow key is pressed. The player should land "on the feet" before being able to jump again.
- Moving Platform
  - There is a moving platform in the map but currently it is not moving. The platform needs to move back and forth to transport the player across platforms.
- Bullets
  - Now that we can move our player freely, it needs a weapon to destroy the enemy! Make the player fire bullets when the space bar key is pressed.
  - Implement a weapon recharging delay such that the player can only fire a bullet every X seconds.
- Destroying Enemy
  - Reduce the Enemy's health when it collides with bullets.
  - Destroy the Enemy when health drops to 0 or below.
- Enemy Fights back
  - Make the enemy fire a bullet every X seconds.
  - The bullet should reduce the player's health when it hits the player.
- Populate the map
  - The map looks kind of empty now. Populate it with more stuff.
    - Suggestions
      - Make the camera follow the player
      - Place platforms
      - Place more enemies
      - Others: Surprise us!!

Extra credits [Optional]

- Object pool
  - Game objects, on creation/destruction, require resources to be allocated/released internally within the game engine. When there is a huge amount of objects being created and destroyed at the same time, performance might suffer, especially on lower resources systems, e.g. mobile phone, tablets.
  - There is a technique where we minimise the creation/destruction calls by creating pools of game objects at the start of the game and destroy them only when they are no longer needed (end of the game/level).
  - Objects are taken out from the pool when needed and returned back to the pool when destroyed. Pool objects will be out of the camera view, hidden and disabled.
  - Implement this for our bullet game objects.
- Implement winning/game over screens with the ability to restart the game
- And any more you can think of! Remember, you are making a game!

Task #3

MAKING IT FUN!

- Jumping around shooting stationary enemies, BORING!
- Design and implement new game play features to make the game fun.

Additional Submission Reminder:

Create a README text file. Kindly put in your full name and school. List down any extra credit features attempted as well as what the new gameplay feature is from Task #3. Explain why your feature makes the game a whole lot more fun. Zip up the README text file and project folder with all of your changes. Submit the zip file. Thanks for applying and do enjoy this exercise. We hope you get to learn something from it. Good luck!

....

END OF TEST