**ASSESSMENT FRONT COVER**

|  |  |
| --- | --- |
| Session | 2018-2019 |

|  |  |
| --- | --- |
| Unit Title | Game Programming Fundamentals and Developing Software: Introduction |
| Unit Code | HH57 34 and H173 34 |
| Assessment No | 1 |
| Validated by IV for current session Yes | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Student’s Name | | Brandon Robson | | |
| Course | | HNC Computer Game Development | | |
| Class | B | | Date |  |
| Student declaration – I agree that all work submitted is my own, or in the case of a group submission, any contributions attributed to me are accurate. | | | | |
| Student’s Signature | | Brandon Robson | | |

ASSESSMENT CONDITIONS (Circle)

|  |  |  |
| --- | --- | --- |
| Closed book | Proforma/template | Case Study |
| Open book | Role play | Project |
| Other(describe): | | |

|  |  |
| --- | --- |
| Time allowed |  |
| Resources required If applicable |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Result \* | Attempt | Date | Lecturer  Signature | IV sign and date (if in sample) |
|  |  |  |  |  |

Pass /Resubmit (previous attempts must be retained)/ Resit /Fail (no further opportunities/attempts for assessment are available).

Feedback to student:

Student Comments:

Remediation Feedback:

IV/Cross-Marker Comments (if in sample)

Agree with assessment decision Yes / No

Do not agree with assessment decision Yes / No

Comments back to Assessor:

Actions for course team:

Assessment 1

## Assessment task instructions

You will be given a “Technical Guide” containing the following:

* Statement of Standards
* Requirements specification
* Detailed design (top level, detailed, and interface design)
* Test strategy
* Test plan

You are required to submit:

* Source code to meet the requirements, standards, and designs
* A completed technical guide, including:
  + The Statement of Standards you were given
  + The Requirements specification you were given
  + The Detailed design you were given
  + The Test strategy you were given
  + A completed version of the test plan you were given

**Assessment conditions:**

The assessment is open-book and you are encouraged to refer to study notes, research online, and examine previously completed code. You are required to complete this work under supervised conditions so that your assessor can authenticate your work.

# Statement of standards

## Entity identifiers

Identifiers of variables, methods etc should:

* be concise but meaningful.
* represent the purpose of the data stored.
* use ‘camelCase’ for all variables.
* use ‘PascalCase’ for methods.
* use local scope where possible, and member scope only when necessary. Do not use glocal scope / global namespace.
* not be specified as static, protected or public unless absolutely necessary.
* be created if needed, to avoid unexplained ‘magic numbers’ in your code.

## Code style

Code statements should:

* each be on their own line, unless the line would become excessively long
* be indented further in each code block , as per the Allman style.

Empty lines of ‘whitespace’:

* must be used before each new method/function.
* Can be used to separate ‘phrases’ of code if you wish.

## Comments

You must use comments within your code:

* immediately before each method/function, to describe that function’s purpose.
* To describe the purpose or operation of some of the lines or sections in the code that are not immediately obvious.

# Requirements Specification

You are required to complete the code for a “Pong” clone game. While doing this, you should use: sequence, selection, variables (properties), iteration (loop) statements and methods with parameter passing.

### Functional Requirements

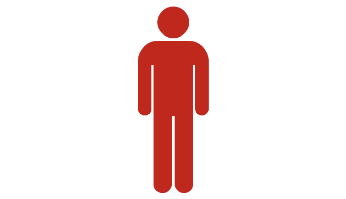
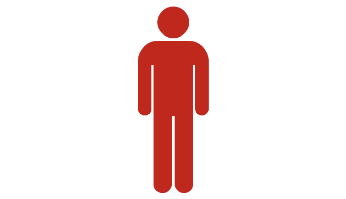
1. At the start of the game, both players should have zero points.
2. At the start of the game, the player’s bats should begin at the middle left (for player one) and middle right (for player two) of the screen.
3. During the match, the two bats may independently move up and/or down if the player presses the corresponding key on the keyboard.
4. If a bat reaches the top or bottom of the screen, it should not be able to move any further in that direction.
5. At the start of the game, the ball should appear in the centre of the screen.
6. At the start of the game, the ball should have a velocity upwards and to the right.
7. If the ball hits the top or bottom of the screen, it should reverse vertical direction.
8. If the ball collides with the left bat, horizontally it will start moving to the right.
9. If the ball collides with the right bat, horizontally it will start moving to the left.
10. If the ball reaches the left edge of the screen, player 2 will score a point.
11. If the ball reaches the right edge of the screen, player 1 will score a point.
12. After scoring a point, the ball will return to the centre of the screen, and move diagonally toward the other player.
13. Player 1’s score will be represented by up to 5 copies of the ball image in the top left corner of the screen.
14. Player 2’s score will be represented by up to 5 copies of the ball image in the top right corner of the screen.
15. If either player scores 5 points, the game ends by stopping the ball movement.

### Non-Functional Requirements

1. The game must use 2d graphics.
2. The game must have sound effects for collisions, and scoring points.
3. It must be a two-player game.
4. You should use a game Engine, for example Unity Engine.
5. The system must operate correctly, as defined by the inputs and outputs in the test document.
6. The product must be delivered on time, as defined by your given hand in date.
7. The codebase must be maintainable and understandable, as defined in the statement of standards.

## Top level design

The following shows the use cases for the system. Note that although bat/wall collision always occurs as a result of moving the bat, the bat/ball collision does not require the bat to actually move (sometimes the ball moves into the bat instead).



<extend>

<extend>

<include>

<extend>

Player

Physics engine

<extend>

The following is a class diagram for the system, showing required variables and methods. Local variables are left to the programmer to determine as required.

|  |
| --- |
| BallScript : MonoBehaviour |
| -startSpeed : Vector2 = (15, 15)  -topBoundary : float = 4.0  -bottomBoundary : float = -4.0  -leftBoundary : float = -8.0  -rightBoundary : float = 8.0  -hitSound : AudioClip  -scoreSound : AudioClip  -body : Rigidbody2D  -soundSource : AudioSource  -player1Score : int = 0  -player2Score : int = 0 |
| -OnGUI() : void  -OnCollisionEnter2D(Collision2d collision) : void  -Start() : void  -SpawnBall() : void  -Update() : void |

|  |
| --- |
| BatMovement : MonoBehaviour |
| -upKey : KeyCode = KeyCode.W  -downKey : KeyCode = KeyCode.S  -batSpeed : Vector2  -topBoundary : float  -bottomBoundary : float  -batBody : Rigidbody2D |
| -Start() : void  -Update() : void |

## Detailed design

## BallScript::Update

The following flowchart shows the process flow through each frame of update. Note that the condition on the right loops backwards, which indicates iteration.

Passed top

boundary?

Passed right

Boundary?

Set vertical speed to negative

Set vertical speed to positive

Player 1 score increases

yes

no

Passed bottom

boundary?

no

yes

Passed left

Boundary?

no

no

yes

Player 2 score increases

Play sound effect

Spawn Ball moving left

Play sound effect

Spawn Ball moving right

Anyone scored

5 points?

no

Destroy the ball’s rigidbody

## BallScript::SpawnBall

Inputs:

integer direction, which should be +1 for moving right or -1 for moving left.

Outputs: none

Pseudocode:

Assign new co-ordinates to ball of 0,0,0

Assign a velocity of startspeed multiplied by direction

**BallScript::Start**

Inputs: none

Outputs: none

Pseudocode:

Get AudioSource component and assign it to soundSource

Get RigidBody2D component and assign it to body

Spawn Ball moving right

**BallScript::OnCollisionEnter2D**

Inputs: none

Outputs: none

Pseudocode:

Play the hitSound audio clip through the soundSource

**BallScript::OnGUI**

Inputs: none

Outputs: none

Pseudocode:

Get texture from SpriteRenderer component and store a reference to it

Set starting X co-ordinate to 15 pixels

Set spacing to 15 pixels

REPEAT

Draw ball sprite texture in rectangle according to spacing

UNTIL we draw as many balls as player 1’s score

Change spacing to -15 pixels

Change starting X co-ordinate to Screen width - 15 pixels

REPEAT

Draw ball sprite texture in rectangle according to spacing

UNTIL we draw as many balls as player 2’s score

## Interface Design

The following annotated screen layouts should indicate how the game interface will appear.

**Main game screen**

1

3

2

4

1: This is where up to 5 ball textures will be drawn as player 1 scores points

2: This is where up to 5 ball textures will be drawn as player 2 scores points.

3: The ball is shown in its spawn point after a point has been scored.

4: The bats are shown at the vertical edges of the play area.

# Test strategy

We will use White Box Testing methods (where we have access to the code we are testing) to perform Development Testing.

Where possible, use a top down approach, employing stub functions (which may be empty, or use simple logging to show they have been called) so that higher level modules can be tested even before lower level modules are fully written.

# Test log sheet

Run your program and complete the last 2 columns of this table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Candidate |  | | Project title |  | |
| Module / Routine | Test case | Test data | Expected result | Actual result | Comments |
| BallScript::Start | Start game | Press play button | Ball begins moving diagonally upwards and to the right | Ball begins moving diagonally upwards and to the right | By default the ball always moves to the top-right |
| BallScript::Update | Ball hitting top | Play until ball hits top of screen | Ball reflects off the top of the screen | Ball reflects off the top of the screen |  |
| BallScript::Update | Ball hitting bottom | Play until ball hits bottom of screen | Ball reflects off the bottom of the screen | Ball reflects off the bottom of the screen |  |
| BallScript::Update | Ball hitting left edge | Play until ball hits left edge of screen | Player 1 gains a point, sound plays, ball returns to centre and moves diagonally right | Player 1 gains a point, sound plays, ball returns to centre and moves diagonally right | Ball texture appears above Player 2 |
| BallScript::Update | Ball hitting right edge | Play until ball hits right edge of screen | Player 2 gains a point, sound plays, ball returns to centre and moves diagonally left | Player 2 gains a point, sound plays, ball returns to centre and moves diagonally left | Ball texture appears above Player 2 |
| BallScript::Update | Player 1 wins | Play so that player 1 gets 5 points before player 2 | Play will end | Play will end (Ball will stop moving) | The ball’s rigidbody is removed thus it stops. 5 ball textures are displayed above Player 1 |
| BallScript::Update | Player 2 wins | Play so that player 2 gets 5 points before player 1 | Play will end | Play will end (Ball will stop moving) | The ball’s rigidbody is removed thus it stops. 5 ball textures are displayed above Player 2 |
| BallScript::  OnCollisionEnter2D | Player 1 bat hits ball | Move player 1 bat to hit ball | Sound plays, ball bounces off | Sound plays, ball bounces off | ‘Hit’ sound effect plays |
| BallScript::  OnCollisionEnter2D | Player 2 bat hits ball | Move player 2 bat to hit ball | Sound plays, ball bounces off | Sound plays, ball bounces off | ‘Hit sound effect plays’ |

# Assessment task 1

|  |  |
| --- | --- |
| Candidate’s name | Brandon Robson |
| Assessor’s name |  |
| Date |  |

**Assessment Evidence Requirements checklist**

|  |  |
| --- | --- |
| Game Programming Fundamentals Outcome 1 Evidence requirement: produce suitable segments of program code | Achieved |
| Variables and data types |  |
| Operators: *e.g.* Arithmetic, Logical, Assignment, relational |  |
| Sequence |  |
| Selection: *e.g.* Conditional statements, Case statements |  |
| Iteration: *e.g.* Loops such as FOR, FOREACH, WHILE, DO WHILE |  |
| Data Structures: *e.g.* Arrays, Lists, Structs, Classes |  |
| Functions |  |
| Parameter passing |  |
| Input/Output: *e.g.* Keyboard, Mouse, Gamepad, Headset, Speakers |  |
| Implementation of digital media within a game environment: *e.g.* 2D graphics such as JPEG, PNG; 3D models and/or animations; Audio files (WAV, MP3 etc); |  |
| Comments |  |

|  |  |
| --- | --- |
| Developing Software Introduction Outcome 1 Evidence requirement: write error free code to carry out at least two different tasks from a given design | Achieved |
| One example of selection (if statement or case statement) |  |
| One example of iteration (for loop, repeat until, do while) |  |
| At least two variables of different data types |  |
| determine the scope (global or local) and use of inbuilt functions  *we here interpret this phrase to apply to usage of namespaces, for example whether to use global namespace aliases or not.* |  |
| a user defined function which will include parameter passing |  |

|  |  |
| --- | --- |
| Developing Software Introduction Outcome 2 Evidence requirement: statement of good practice… expected coding standard acceptable | Achieved |
| naming conventions for variables and functions |  |
| use of global and local variables  *ie. Learner has used global / local as specified in the design documents. It is not necessary to use both unless the design document stipulates it.* |  |
| use of indentation |  |
| comments. (“include comments as internal documentation”) |  |

|  |  |
| --- | --- |
| Developing Software Introduction Outcome 1 and 2 Evidence requirements: technical guide | Achieved |
| description of the system, detailed design (both assessor provided) |  |
| a structured listing (commented and well laid out code) |  |
| completed test logs. (demonstrates that the code meets the desired result of the given test plan) |  |