



PROGRESS OUTCOME 5

Pirate Game

Annotation

Tua is able to use algorithmic thinking to develop two different algorithms to solve a specific problem. He uses logical reasoning to evaluate the relative efficiencies of the algorithms for program performance.

He demonstrates that he can operate within a text-based programming environment to create a program to implement an algorithm. His program reveals an understanding of:

- iteration (the step event repeats constantly)
- selection through conditional logic (if player_coins <=0)
- global variables (player_coins)
- derived values (room_height and room_width)
- random numbers
- functions and parameters (instance_create).

Tua provides documentation for the program through the use of comments. He also demonstrates that he can apply iterative development and testing to identify a missing step in a program before moving on to the next part of the program.

Background

The students in Tua's class are learning to program in a text-based programming environment. They have developed and tested several programs that use variables, functions, random numbers, and derived values. They have practised the decomposition strategy of breaking down a program into functional requirements, breaking each requirement down further into an algorithm, and then translating this into programming code. They have also explored iterative development by testing each part of their program before moving on to the next requirement.

Task

The students are given a game called The Pirate Game in a text-based programming environment (GameMaker: Studio). The game has been deliberately "broken" by their teacher, Mr Walker - code is missing that is necessary for the game to run. The students play the game to determine its rules and what has been broken.

The students are asked to develop a set of functional requirements for the game that cover its objectives and how a player would interact with the game.

Next, they are asked to develop two algorithms for the missing code and to evaluate them in terms of their effect on the program's performance. They are required to program, test, and debug the algorithm that they decide is the most efficient in order to ensure the game functions as intended.

Student response

After playing The Pirate Game, Tua develops the following set of functional requirements.

Functional Requirements for the Pirate Game

- If the pirate touches the player it should destroy itself and subtract a coin.
- When all coins are lost the game should end.
- When a pirate reaches the edge of the screen it will respawn in a random location on the Y axis.
- Pirates will spawn in a random location once destroyed.
- WASD keys are used for player movement.
- Sound button turns music on and off (toggle).
- Player can choose a sprite.

He realises that the bug in the game is that the game never ends. This is because there is no program code to determine whether all the coins are lost and what should happen when they are. He develops two algorithms to fix the bug in the program.

Algorithm 1 for testing coins

Pirate Object

Forever:

 If player coins less than or equal to zero:
 End game
 If pirate touches player:
 Player coins decrease by one
 Pirate object is destroyed

Algorithm 2 for testing coins

Player Object

When player touches pirate:

 Player coins decrease by one

 If player coins less than or equal to zero:
 End game

 pirate object is destroyed

Mr Walker and Tua discuss the relative efficiencies of Tua's algorithms.

Mr Walker: *Which algorithm would be better in terms of the use of computing resources and game play?*

Tua: *The first algorithm would make the game laggy because it is constantly testing the coins and collisions on every pirate object. The pirates respawn, so that would mean even more checking. I think my second algorithm is better because it focuses on the player collision event and there is only one player object. And it only checks the number of coins when the player collides with a pirate. This is better because the calculations and decisions happen only when they need to, not all the time.*

Tua implements his second algorithm by writing code in GML. When testing his code, he finds there is still a problem – pirates are not respawning in the game, so the game play cannot continue after a pirate collides with the player.

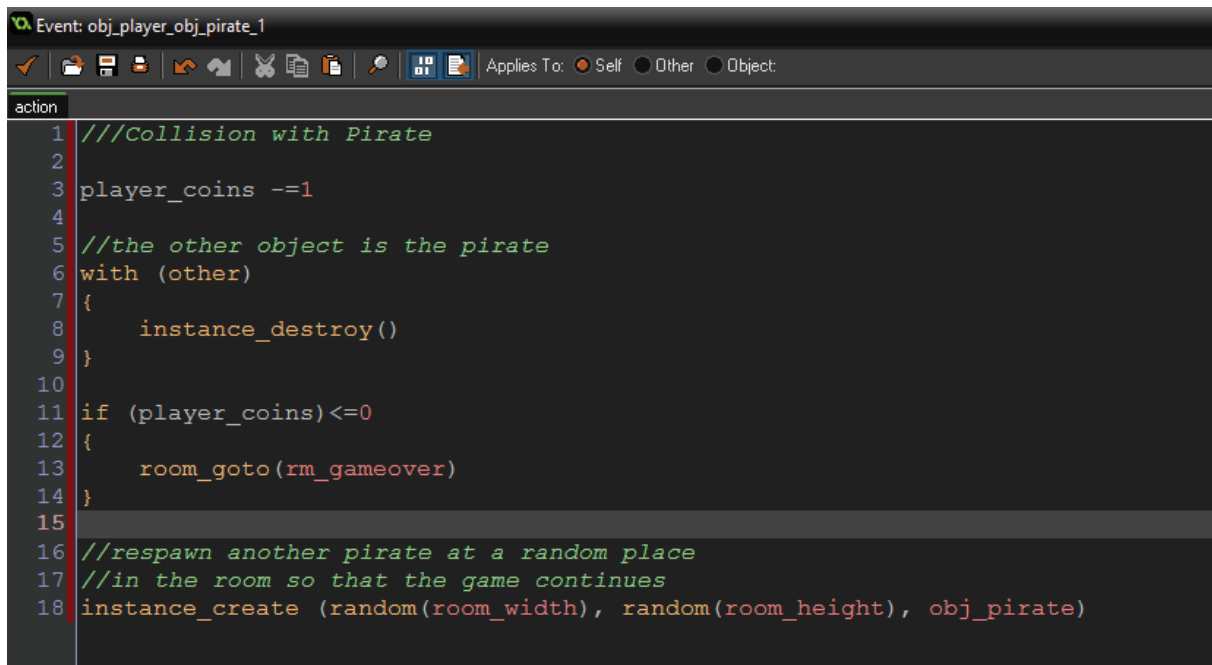
Mr Walker and Tua discuss the bug in his program.

Tua: *The game stops after one collision with a pirate. The pirate doesn't respawn.*

Mr Walker: *What should happen next to continue game play?*

Tua: *I need to make a pirate respawn in the room at a random location.*

Tua adds the function to create another pirate at a random location within the game screen. He then tests his final version to ensure that it allows the game play to continue.



```
Event: obj_player_obj_pirate_1
action
1  ///Collision with Pirate
2
3  player_coins -=1
4
5  ///the other object is the pirate
6  with (other)
7  {
8      instance_destroy()
9  }
10
11 if (player_coins)<=0
12 {
13     room_goto (rm_gameover)
14 }
15
16 ///respawn another pirate at a random place
17 ///in the room so that the game continues
18 instance_create (random(room_width), random(room_height), obj_pirate)
```