CAB202 Assignment 1 Report

Ollie Pye - n9703977

11 September 2018

Contents

1 Introduction

The screen height must be 41 or more characters for the blocks to fit on the screen.

A major bug is that the player can go in through the sides of blocks and when falling next to a moving block the player gets stuck on the side.

Compiling and running this program has been tested successfully on Windows, Mac and Linux.

2 Implementation Summary

1. **Display Screen** Fully implemented
2. **Player size and initial position** Fully implemented
3. **Block size and spacing** Fully implemented
4. **Blocks are randomly positioned** Fully implemented
5. **Columns and block types** Fully implemented
6. **Player Movement** Fully implemented
7. **Treasure** Fully implemented
8. **Basic game mechanics** Fully implemented
9. **Block width and movement** Fully implemented
10. **Advanced player motion** Fully implemented
11. **Ballistic motion** Partially implemented
12. **Player random re-spawning** Fully implemented
13. **Treasure animation** Fully implemented
14. **Player movement animation** Fully implemented
15. **Player respawn animation** Not implemented
16. **Minimal duplication** Fully implemented
17. **Arrays or pointers** Fully implemented
18. **Functions** Mostly implemented

19. Parameters Fully implemented

3 Testing

Test of Specific Functionality	**Test Setup**	**Expected Result**	**Actual Result**
Display Screen: Display is 5 characters high, at the top of screen, always visible and displays score, time, lives and student number.	Run game through different scenarios (moving, touch forbidden block, fall through boundaries, get treasure).	Display should always be visible and contain the correct information regardless of the game situation. Time, lives and score should update dependent on game situation.	As expected
Player size: Player dimensions are 3x3. Advanced functionality is implemented so that the player spawns randomly. As such, there is no 'starting block'.	Run game and ensure player remains 3x3 through various scenarios (jumping, moving, falling).	Player will remain 3x3 throughout game regardless of situation.	As expected
Block size: All blocks are 2 characters high and at least sprite height + 2 away from other blocks. Advanced functionality is implemented for block width.	Run game and inspect blocks.	Blocks are all aligned, spaced and only ever 2 characters high.	As expected
Block type: At least 1 safe block per column and 2 forbidden on the screen.	Run game, inspect blocks.	Larger number of safe blocks than forbidden. Blocks identified with 'x' and '=' characters.	As expected
Block random positioning: Blocks have no consistent observable pattern.	Run the game, go through various scenarios and inspect blocks.	Blocks have no recognisable pattern. There is no overlap, blocks are random. The location of blocks for the next game are not predictable.	As expected

Player movement: ‘a’ and ‘d’ keys move player left or right, player does not overlap display. Player free falls and no lateral movement occurs when falling.	Run the game, moving the player left and right. Move off block to observe fall. When falling attempt to move laterally. Attempt to jump through display screen.	Player moves left and right only when on block, not while free falling. Player does not overlap display screen. Player free falls when off a block.	As expected
Treasure: Treasure moves along the bottom, reversing direction when reaching screen edge. Motion can be paused by pressing ‘t’. Two lives are added when player collides with treasure, and the treasure disappears and returns to the left edge. Is no larger than player and is distinct image.	Run game long enough for treasure to move across screen several times. Press ‘t’ to test pause in motion at various points along the screen (edge, middle). Collide player and treasure.	Treasure moves between the screen edge, stops/starts moving when ‘t’ is pressed, disappears and returns to bottom left corner when player collides and lives increase.	As expected
Basic mechanics: Player starting with 10 lives, scores 1 point when landing on safe block, dies when falling off screen or hits forbidden block. Display screen of score and time is present when all lives are lost, with the option of restarting (r) or quitting (q).	Run game until all lives are lost. Lose lives in various ways (by falling down the bottom and landing on forbidden blocks). Test both reset and quit.	Score is added as player goes from block to block. Life is lost when player touches forbidden block and no score is added. Life lost when player falls down the bottom. When lives is 0, quit screen is displayed. Both reset and quit options only work with assigned keys, and other keys have no influence.	As expected

Block width and movement: Random block sizes with centre rows have opposite motion. Blocks reappear on opposite side of display when they move off the screen. Blocks on the same row move at the same speed.	Run the game. Observe the blocks dynamics.	As blocks move off the display they reappear on the opposite side. Blocks in same row travel at the same speed and do not over lap due to inconsistent speeds. Top and bottom row remain stationary. Each row moves at different speed and direction to the one above and below.	As expected
Advance player motion: Pressing 'a' and 'd' gives the player horizontal velocity. 'w' allows player to jump and nothing else until landing. Player adopts block velocity. Player loses life when block takes it off screen.	Run game and include all movement operations. Move to various blocks and determine if speed of block is adopted. Allow moving block to take player off screen.	Pressing 'a' and 'd' gives the player velocity in that direction and increases if you keep pressing. To stop/slow down, the opposite key is pressed. Blocks other than those on the top and bottom row move, and disappear/reappear when going off the edge of the screen. Player dies if it is carried off screen by blocks. Pressing 'w' will make the player jump, and pressing anything else will not affect the player's motion until it has landed again. Player will adopt the speed of moving blocks.	As expected
Ballistic Motion: When on a block, gravity is eliminated.	Run the game and test whether landing on safe blocks stops the player from falling. Try on multiple blocks (i.e. moving/stationary).	This is only partially implemented. Gravity is constantly acting downwards on player. The player does not fall through blocks in any situation, however. All other aspects of ballistic motion are not implemented and as such are not tested.	Partial as expected.

Random Respawn: Player spawning on a random safe block after death.	Run the game and die/respawn via different pathways (i.e. falling on forbidden block/colliding with treasure/fall off screen) and observe the location of the player respawning.	When the player dies or touches the treasure, it should respawn on a random safeblock. This is also expected when the game first starts or restarts. It should not land on forbidden block or land between two blocks.	As expected

4 Implementation of advanced graphics

The treasure animation and player image changes have been implemented. To observe these implementations, run the game and observe the motion of the treasure. When the game is in progress, the treasure flashes between two symbol types '$' and '#'. This animation is consistent regardless of the game situation (unless game over). The player animation occurs when the player is moving or falling. Run game and observe the image of the player when moving in different situations. The player has a separate image for falling/jumping, moving and when stationary on the screen.

The player animation on respawn has not been implemented.

5 Implementation of advanced coding practices

If pieces of code needed to be used more than once, they were placed in a function to reduce duplication. Furthermore, to ensure readability, function and variable names that described their purpose were used. Reduction of duplication and use of practical function and variable names ensures that the reader can easily understand what the code is doing. The variable/function role is implicitly stated in it's name, and the user is not confronted with mountains of repeated code that would be distracting.

Arrays and pointers have been used effectively throughout this program. All of the blocks are stored in a single array, making then easily accessible and convenient to work with. Pointers have been used extensively. Pointers were used for changing the speed, image, location etc. of sprites. They were used as it does not require the duplication of memory. Instead the change is being made at the location of the memory. Furthermore, sprite_init was used to reduce memory usage. This improves efficiency of the program.

This program is made entirely of functions. There are one or two functions however, that exceed the 20 line limit. The use of functions reduces duplications and makes the code easier to manipulate/update in the future. Functions also make the code more dynamic as they can be easily implemented within the program making them effective tools for programmers.

This program used parameters in an effort to reduce the number of global variables. The main use of global variables in this program are constants. For values that are dynamic/changing, parameters were used. This is to protect the program. Using global variables that are dynamic increases the risks of bugs, as it is easy for a function to manipulate the global variable without it being obvious to the programmer.