James Moran

Documentation on the Design of the Collision System for, As Well As the Core Design Structure for, Our Game

2016

Contents

[Introduction 2](#_Toc465417611)

[Initial Ideas 2](#_Toc465417612)

[Game Flowchart 3](#_Toc465417613)

[Level Generation 4](#_Toc465417614)

[Display Level 5](#_Toc465417615)

[Top Level Game Class Structure 5](#_Toc465417616)

[Tasks 6](#_Toc465417617)

[Part 1 Reflection 6](#_Toc465417618)

[Top-Level Design Overview 7](#_Toc465417619)

[Consideration of the Required Classes 7](#_Toc465417620)

[Stepwise Refinement – First Pass 8](#_Toc465417621)

[Refinement of the ‘Check for collision’ Sub-Process 8](#_Toc465417622)

[Refinement of the ‘Handle the Collision’ Sub-Process 9](#_Toc465417623)

[Refinement of the ‘BounceOffWall’ function 9](#_Toc465417624)

[UML Class Structure 10](#_Toc465417625)

[Part 2 Reflection 11](#_Toc465417626)

[Appendix 12](#_Toc465417627)

[Figure 1: Chat-Log Whilst Putting Together the First Game-Flowchart (as a group) 12](#_Toc465417628)

Part 1

# Introduction

The game that our team is putting together; is a 2D-Platform-Based-Fixed-Perspective-Evasion-Game, where the Player must escape from a room by finding a key whilst avoiding contact with Enemies. When the Player finds the Key to escape the room, they can open the Door to leave the room, by approaching it. The Player and any Enemies will bounce away from the Door (unless the Player has the Key) and any Walls. Only the Player can interact with the Key (by picking it up when they move over it), as well as all other Entities within the game. Enemies can only interact with the Player, any Wall Segments and the Door, if the Player collides with an Enemy, or an Enemy collides with the Player; the Player will lose a life and respawn at the Spawn Point within the current level. The Player has 3 lives (including their current life at the start), should the Player lose these lives, before completing the level; the game is over for them and so, they must try again to continue onwards through the available levels (10 in total). The game will utilise SDL 2.0.

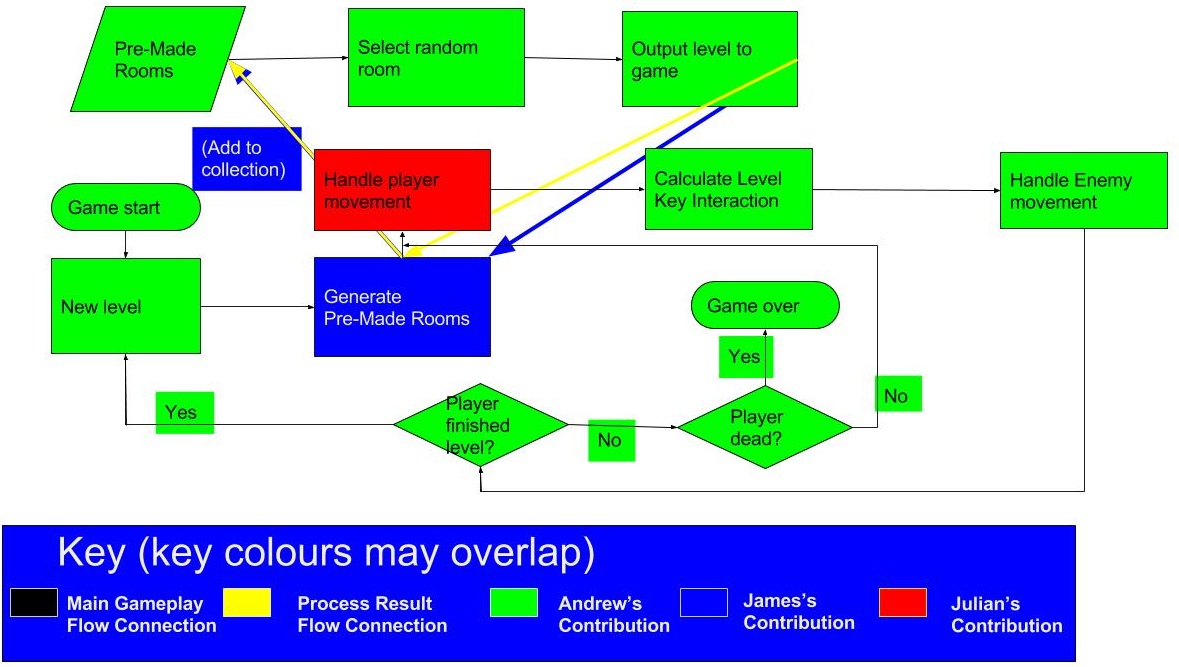
# Initial Ideas

Our initial ideas are as follows:



From these ideas, we decided upon a 2D-Platform Game, where the Player would have to escape from a room by finding a key, with Enemies patrolling along pre-set routes. This is detailed in the following section:

# Game Flowchart

The flow of our game is as detailed below:

Edit: In the latest version, the game flowchart has received a fair few alterations, as shown below:

(See Fig. 1 for a chat log between team members whilst we were putting this flow chart together.)

From this flowchart: Edit 1: I created an altered version for such, as shown to the left:

There is no game menu for our game: When execution of the game begins; the first level is generated and shown to the Player, whom can then start playing after the level is generated, an overview of ‘Level Generation’ is shown in the next section:

## Level Generation

Refinement of this step in playing the game, is as follows (Edit 1: Altered the flow chart to note selecting a room from the pre-made room set):

After this diagram, is the process for displaying the level. The pseudo-code for such is in the next section (below):

## Display Level

1. Generate a new level structure
   1. Given pre-set dimensions, create a level structure with such extents (as many tiles horizontally as for the width given and as many tiles vertically as for the height given, this hence; becomes a grid of tiles).
   2. When each tile is generated, assign it an EntityID, for checking what to display to the Player, in the next step.
2. For each tile in this grid of tiles (from the first row (0,0) to last the row (width, height)):
   1. If the EntityID of this tile is that of ‘Wall’…
      1. Display a wall tile at the current location and return to the top level of this step, for the next tile.
   2. If the EntityID of this tile is that of ‘Key’…
      1. Display a key pick-up tile at the current location and return to the top level of this step, for the next tile.
   3. If the EntityID of this tile is that of ‘Door’…
      1. Display the door tile for this level, at the current location and return to the top level of this step, for the next tile.
   4. If the EntityID of this tile is that of not equal to any of the above tile-identities…
      1. Display no tile in this location (utilising a sparse-array/vector for displaying the level).

# Top Level Game Class Structure

# Tasks

The list of tasks that we put together as a team; are listed below:



As detailed in the above list; I was delegated the task of handling collision between the Player and any Enemies.

# Part 1 Reflection

In reflection of part 1, overall; we, as a team, could come together and agree upon which tasks certain people should undertake.

From initial ideas for the game, to determining which mechanics our game would have to finalising these initial thoughts, for a game that is a 2D-Platform-Based-Fixed-Perspective-Evasion-Game. Moving on from this; we put together a flowchart for the top-most level of our game (cooperatively, putting together a Google Drawing simultaneously).

Then after a delay of approximately one fortnight; we allocated tasks for each team member, with me receiving the task of handling collision in our game (for the Player and any Enemies). With Andrew handling the movement system for the Player and Enemies, whilst Julian is handle the map generation system.

Even so; other problems occurred for our group, such as team members not coming to pre-arranged meetings, miss-matching design patterns, delay in the finalisation of tasks to allocate amongst the group, as well as problems getting hold of certain pieces of software.

Part 2

# Collision System Design Overview

Given what is mentioned in Part 1 of this design document; my initial thoughts for designing a collision system (at a top level), equate to such (see below):

## Consideration of the Required Classes

As well as a Player and Enemy class, classes for the Door, the Key and Walls are also required, in addition; these 5 Entities will inherit either directly or indirectly for an Entity class (with generic member variables for each Entity, such as its bounds and position), plus a character class (which directly inherits from the Entity class), for the Player and Enemy class to inherit from (with a collision mask member; to determine what Entities ‘This Character’ will have interactions with).

Edit: As collision is only going to receive handling on moving game entities; the static components of the level (the Walls, the Key etc.), will not require classes, only the Player and Enemy will require them.

# Stepwise Refinement – First Pass

## Refinement of the ‘Check for collision’ Sub-Process

The first method that comes to my mind; is that of utilising Bounding-Box collision, as this is a sufficient method to check for collision between 2 entities (sufficient in terms of the precision, for this collision-checking method).

Edit 1: The first version of this pseudo-code (describing this sub-process); assumes that the anchor point for Entities (represented by images), is within the absolute centre of that Entity, as this is not the case within SDL (for the origin, from which I presume the anchor point extends from), as the origin is in the top left corner, the pseudo code for this sub-process; has received the following alterations:

1. If this Character’s X-position plus this Character’s width is greater than or equal to the other Entity’s X-position; go to step 1.1, otherwise; go to step 2.
   1. If this Character’s Y-position plus this Character’s height is greater than or equal to the other Entity’s Y-position; there is a collision, so end this sub-process here and pass this result to the super-process, otherwise; go to step 2.
2. If the other Entity’s Y-position plus the other Entity’s height is greater than or equal to this Character’s Y-Position; go to step 2.1, otherwise; go to step 3.
   1. If the other Entity’s X-position plus the other Entity’s width is greater than or equal to this Character’s X-Position; there is a collision, so end this sub-process here and pass this result to the super-process, otherwise; go to step 3.
3. If this Character’s X-position is less than or equal to the other Entity’s X-position plus the other Entity’s width; go to step 3.1, otherwise; go to step 4.
   1. If this Character’s Y-position is less than or equal to the other Entity’s Y-position plus the other Entity’s height; there is a collision, so end this sub-process here and pass this result to the super-process, otherwise; go to step 4.
4. If this Character’s Y-position is less than or equal to the other Entity’s Y-position plus the other Entity’s width; go to step 3.1, otherwise; go to step 4.
   1. If this Character’s Y-position is less than or equal to the other Entity’s Y-position plus the other Entity’s height; there is a collision, so end this sub-process here and pass this result to the super-process, otherwise; there is no collision, so pass this result back to the super process.

Edit 2: The second version of this pseudo-code (describing this sub-process); assumes that ‘This Character’ would interact with ‘Other Entity’ (even if this is not the case). A collision-mask (bit-mask) is used to filter out Entities that This Character would not interact with. This is accounted for in the Class Structure Diagram for this process (go to the ‘UML Class Structure Diagram’ section, to see this diagram).

Edit 3: The logic for this pseudo-code is subject to alteration in implementation.

## Refinement of the ‘Handle the Collision’ Sub-Process

To handle collision with an Entity that ‘This Character’ has collided with, for which, is a valid Entity within This Character’s collision mask, for them to collide with, handle the collision result, as follows:

1. If this Entity is a wall, this Character will bounce off and away from the wall, then end collision handling here, otherwise; go to step 2.
2. If this Entity is the level exit door; go to step 2.1, otherwise; go to step 3.
   1. If this Character is an Enemy, they will bounce off and away from the door, as if it was a wall, then end collision handling here. Otherwise; go to step 2.2.
   2. If this Character is the Player; go to step 2.2.1, otherwise; end collision handling here.
      1. If the Player has the key to the door; this door will open, otherwise; the Player will bounce off and away from the door, as if it was a wall, then end collision handling here.
3. If this Entity is the key to the door (and if this Character is the Player); the Player now has the key and will pick it up, then end collision handling here. Otherwise; go to step 4.
4. If this Entity is an Enemy (and this Character is the Player); the Player will lose a life and respawn at the level’s spawn point, then collision handling ends here. Otherwise; go to step 5.
5. If this Entity is the Player (and this Character is an Enemy); the Player will lose a life and respawn at the level’s spawn point, then collision handling ends here. Even if this is not the case; end collision handling.

## Refinement of the ‘BounceOffWall’ function

ApplyForce(GetDirectionFromEntityToEntity(ThisEntity, ThisCharacter), ThisCharacter);

# UML Class Structure



Entity is the super-class for all Entities, within the scope of this branch for the project, from which any Wall Segments, the Key and the Door, directly inherit from. For the Player and any Enemies though, they inherit from the Character class (which inherits from the Entity class), as there is a collision mask within the Character class for Entities that are of a certain sub-class, of the Character class (would interact with), as well as functions to handle movement for any of these sub-classes. Note also that ECollisionMask is used both for identifying the respective Entity, as well as for the interact-able Entities of a certain Character (for whom this Character should check collision against).

Edit 1: This class structure would not consider the rest of the game. So, I have added an additional page to the class diagram for this game; as is represented in the ‘Top Level Game Class Structure’ segment of this document.

# Part 2 Reflection

In reflection of part 2, of this design document for our game; I believe I have covered all of the aspects for the sub-process delegated to me (from initial analysis of the task for which I am to design the implementation for, to a top-level flow chart for such, to ultimately going deeper into each step of the top-level flow chart and breaking down the sub-process (within this overall sub-process), into pseudo-code, to increase the clarity of how each sub-sub-process is executed within the overall sub-process).

Looking over part 2, at first; I also identified issues in my logic for the design of this system, noting that I have changed my design at each respective point within part 2, for where I have altered the design of this system (see ‘Refinement of the ‘Check for collision’ sub-process’ Edit 1, Edit 2 and Edit 3 section. As well as Edit 1 of the ‘UML Class Structure’ section, for the respective editing to these components of the collision system).

# Appendix

## Figure 1: Chat-Log Whilst Putting Together the First Game-Flowchart (as a group)

Andrew Abraham joined group chat.

**Andrew Abraham**

17:07

Hi, James.

**me**

17:08

oh hi :)

Where is Julian?

**Andrew Abraham**

17:09

Not sure, he'll probably be a few minutes, I guess.

**me**

17:10

Hmm, at any rate, we can start on the ideas flow chart

**Andrew Abraham**

17:11

Ok, I'll see if i can create the draw doc.

**me**

17:12

I understand

Andrew Abraham joined group chat.

**me**

17:13

Ok then **(The next set of messages (Starting with the message below), are from the chat of that draw document)**

**Andrew Abraham**

17:14

Ok, that was simple :)

**me**

17:17

Just a moment, I am looking at the brief

**Andrew Abraham**

17:17

ok

**me**

17:22

Hmm...I'm thinking about the brief for this still

**Andrew Abraham**

17:25

How about we just put out a vague structure first and then itterate upon it in more detail; hopefully Julian will have joined us by the tome we are at least part way through.

**me**

17:26

Ok, that makes sense

**me**

17:34

Hmm, its alright so far

**me**

17:43

To verify; as the standard gameplay flow from one node to another is black, I thought I should note that as...

After generating the Pre-Made Rooms, that should (as a final part of that process), add the resulting rooms that this process generates, to the Pre-Made Rooms set

(So I have put in place a key to distinguish between the two)

**me**

17:48

Also, after a room is chosen from the set of rooms and that room is shown to the Player, then flow should continue past the 'Generate Pre-Made Rooms' node

Would this make sense to you?

**Andrew Abraham**

17:49

I think it does, yes.

**me**

17:51

Ok :)

Then I wonder; would we want the game to query if the Player has finished the level when the level has only begun?

(for the Player\*)

**Andrew Abraham**

17:53

Probably not, but this just a generalisation of what the rest of the code outside of level generation will be.

it need not be 100% accurate beyond "output level to game" for the purposes of this exersise.

**me**

17:57

Right, but will we have further flow charts to define more specific tasks to allocate to members of the group?

(Not tasks of the lowest possible level, but of sufficient specificity to allow a group member to expand upon it the lower level aspects of such)

**Andrew Abraham**

17:57

yes, hopefully if Julian shows up we can allocate the rest of the tasks affter we have completed the first few functions for level generation.

**me**

18:00

upon it, for the\*

Also (from the assessment brief): 'Don't forget to document this process and keep track of how each member contributed and any problems encountered. '

So, I believe we should denote which nodes and other segments people have added to this flow chart

**Andrew Abraham**

18:01

ok

**me**

18:05

Heh, so all bar-1 of the nodes, are courtesy of you so far :D

**Andrew Abraham**

18:06

I'm pretty sure you did a fair amount as well :D

**me**

18:11

Heh

This is simply for whom has initialised the creation of particular components of this flow chart

for this\*

(Hence why I have added a box that surrounds the key, with my colour)

Ah thanks

There :)

**Andrew Abraham**

18:11

Please, give yourself some more credit.

**me**

18:12

Thanks :)

**Andrew Abraham**

18:12

No problem :D

**me**

18:18

I will also create a log of this chat (for future reference), that I will upload to our section of the drive

Ah yes...

You got it :)

**Andrew Abraham**

18:20

Shall we meet up tomorrow to discuss this in further detail?

**me**

18:20

Aye, what time are you thinking of?

**Andrew Abraham**

18:23

I am free any time after 1 pm, shall we say 1:15 in the resource center?

The one on the main floor (I forget the name).

**me**

18:24

Andrew's Learning Resource Centre?

**Andrew Abraham**

18:25

That's the one! :)

**me**

18:25

Alright, we shall meet up there at 13:15 tomorrow then.

**Andrew Abraham**

18:27

Ok, I will let Julian know. :D

**me**

18:28

Indeed, I will take a log of the chat messages for this document at any rate

|  |
| --- |
| **Andrew Abraham**  ok, bye 18:29  Andrew Abraham left group chat.  Andrew Abraham joined group chat.  **me**  18:30  Bye  Andrew Abraham left group chat.  Julian Bath joined group chat.  Julian Bath left group chat. |
|  |