IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 7.39 ⁜ Vol. 11, Issue 4, April 2022

DOI: 10.17148/IJARCCE.2022.114135

A Peculiar Review On 2-D Platformer Game Development

Prabhjot Kaur[1], Manas Jagota[2], Muskan Chopra[3], Nishit Singhal[4], Devansh Singh[5]

[1]Assistant Professor, Indraprastha Engineering College

[2-5]Student, Indraprastha Engineering College

Abstract: The purpose of this thesis is to address design and development issues that have been identified in the platform game genre. The issue at hand stems from the fluctuating curve of interest in the platform-game genre that can be seen from the 1980s to the present day. The problem addressed in this thesis is that modern platform-game developers are prone to overlooking and/or deprioritizing important design and gameplay elements found in previous popular games in the genre.

This thesis concludes with a comprehensive presentation of the Implementation categories, complete with complexity examples for each category and a complete game. The findings of this thesis should be useful in small-scale, independent, or academic game projects in terms of design, decision-making, prioritisation, and time management.

Keywords: game development, game design, platform games, software complexity, framework, analysis, prototype.

I. INTRODUCTION

The purpose of this chapter is to provide a general overview of the thesis' background and significance. In addition, this chapter will present the thesis problem formulation and hypothesis, as well as the intended target group and the order of the following chapters.

A. Background:

A platform game, also known as a platformer, is a game in which the player must move and jump between obstacles and objects that serve as platforms. Because it is frequently combined with other game genres such as Shooters and R.P. Games, the genre cannot be fully considered a standalone genre.

It's worth noting that the recent resurgence of interest in the platform game genre appears to coincide with the release of these games, as the majority of them are remakes of some of the best-selling 2D platform games from the 1980s and 1990s.

B. Problem formulation:

The purpose of this thesis is to expand on D. Boutros' reasoning regarding the reasons for the decline in interest in the platform genre. In fact, the problem formulation in this thesis can be considered, for all intents and purposes, equivalent to Boutros' conclusions. As a result, the thesis's problem formulation is based on the hypothesis that critical design elements - which may contribute to a platformer's success rate - are frequently overlooked or deprioritized during the development process. D. Boutros makes the following conclusion, which will be the thesis's main problem:

- Effective design principles from the past are frequently overlooked or forgotten in modern platform game development.
- Lack of familiarity with common design elements and features from previous successful platform games.
- During the development process, there is a lack of understanding of the time consumption, complexity, and prioritization of these components.

Hypothesis: In order to address this issue, we must look at the limitations of older games as well as the software that is used with newer versions and gaming engines.

We use a Unity game that we created and compare it to older Platformer games.

C. Limitations:

Due to the time constraints of this thesis, it is important to note that the final prototype used to measure complexity is rather small-scale. In the case of more complex projects, this does not necessarily imply that the data derived from the measurement will be inaccurate. However, the possibility of different results depending on the project's scale must be

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 7.39 ⁜ Vol. 11, Issue 4, April 2022

DOI: 10.17148/IJARCCE.2022.114135

taken into account. Furthermore, it should be noted that the final prototype did not include all of the subcategories presented within the Implementation categories.

D. Software Complexity:

The complexity of a working game is influenced by a variety of factors, including:

- **Lines of Code:** This is perhaps the most basic metric for determining the size of a method or class. In comparison to earlier times, the number of lines of code used in our game is extremely low. Together, Unity and Visual Studio have drastically reduced the number of lines of code required to create inbuilt functions. The codes are also simpler to comprehend for non-developers.
- **Cyclomatic Complexity:** It is a rather complicated metric devised by Thomas J. McCabe for assessing a program's logical strength. This metric will be used in this study to determine the number of decisions made within a given class or method. The inclusion of this metric is motivated by the fact that the number of decisions made within a class or method obviously affects the amount of time spent programming.
- **Depth of Inheritance:** This metric "indicates the number of class definitions that extend to the root of the class hierarchy," according to its definition. According to L.H Rosenberg et al. [10], a high depth of inheritance within a class increases design complexity while also increasing the potential for reuse.
- **Class Coupling:** This metric, also known as "object coupling," assesses the relationship between a given class and other distinct classes. Variables, parameters, return types, method calls, base classes, and interfaces are all examples of how two or more classes can be linked.
- **Maintainability Index:** The final metric in this thesis evaluation is an index value that represents the ease with which code can be maintained. In order to calculate its index value, this metric uses lines of code and cyclomatic complexity metrics. Any code with an index value greater than twenty is considered to be maintainable.

II. COMPARISON OF GAMES

A. Method:

This chapter will go over the methods used in this thesis in great detail. This thesis was completed in two steps: a theoretical analysis that included the creation of a light implementation framework, and a practical, iterative prototype development that included complexity measurements.

A.1 Implementation Categories

Before we can create a light framework for developing a platform game with the goal of measuring the complexity of its various design and development-related categories, we must first figure out what they are.

We divided all of the platform game components we found relevant into three main categories based on these already identified platform game components and additional research in the area: Level components, Avatar mechanics, and Level visuals. These three categories will be referred to as Implementation categories in this thesis.

A.1.1 Components at the Level

We named the first implementation category Level components. G. Smith et al. identified level-associated components, which fall into this category. The components within each implementation category will be referred to as subcategories in this thesis. The following are the subcategories within Level components that are defined and motivated:

- **Platform –** A platform is any object that the player can move across, jump on, or navigate on. Because it is the very core element of a platform game, the motivation for including these 11 components is obvious. Even though platforms are required for a platform game, we found it interesting to investigate the actual complexity of this functionality because it is likely to be the highest prioritized functionality at the start of any platform game development, alongside player mechanics.
- **Obstacles –** This component is defined by Smith et al. as "any object capable of causing damage to the player." This subcategory will include all enemies, projectiles, and gaps that cause damage or death to the player in this thesis.
- **Movement aids –** Any object that, in addition to the player's own mechanics, aids the player's ability to move. Trampolines, ladders, and ropes are examples of such items. We believe this subcategory is an excellent example of components that are frequently overlooked. As a result, it's crucial to look into the actual complexity and time commitment that a feature like this might entail.
- **Triggers –** Smith et al. define triggers as "interactive objects that the player can use to change the stage of the level, or even game rules such as physics." This subcategory, we believe, is another good example of functionality that isn't always prioritized.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 7.39 ⁜ Vol. 11, Issue 4, April 2022

DOI: 10.17148/IJARCCE.2022.114135

- **Collectable items –** This last subcategory includes any game item that provides a reward. Coins, extra lives, and power-ups are examples of such items. The complexity of this subcategory is also considered to be of great interest, as collectible items can improve gameplay in a variety of ways, but they may not be given the attention they deserve.

A.1.2 Avatar Mechanics:

Avatar mechanics is the name we gave to the second implementation category. The term "avatar" simply refers to the game's player character. The reason for this definition is that the term "player" is sometimes used to refer to the person who is playing the game rather than the in-game character in some studies. The player in this thesis refers to the actual in-game character, not the person who is playing the game. Nonetheless, we have chosen to use the term "avatar" in our definition. Items that can be collected – Any game item that provides some sort of reward falls into the last subcategory. Coins, extra lives, and power-ups are examples of such items. This subcategory's complexity is also regarded as significant.

The following subcategories are defined and motivated within this category:

- **Collision –** This subcategory includes all functionality related to avatar-environment collisions, such as platforms and obstacles. This subcategory was not found in any previous research, possibly because it is solely from the perspective of a developer rather than from the perspective of a designer. However, from the perspective of a developer, this subcategory is critical because this functionality may necessitate a significant amount of time and effort.
- **The fundamentals of mechanics –** In addition to D. Boutros' discussion of player controls, this subcategory encompasses the functionality that represents all of the avatar's basic mechanics, such as jumping and moving.
- **Additional Avatar Mechanics –** The final subcategory will cover any additional Avatar Mechanics. The desire to create this subcategory stems from the vast array of options available for the avatar. A platform game, as stated in the introduction, may or may not include elements from other game genres such as shooters. With this in mind, we can deduce that avatar mechanics could include a wide range of activities such as hitting, shooting, and crawling. As a result, any avatar mechanics other than the more traditional ones mentioned in the previous subcategory should be placed here.

A.1.3 Level Visuals:

The third and final implementation category, Level Visuals, encompasses all graphical and audio components. D. Boutros' observation of recurring first-level graphic themes among several of the most popular platform games to date inspired this implementation category. Furthermore, it would be highly unreasonable for a developer to ignore the game's aesthetic aspects. We assume in this thesis that well-designed functionality may not be recognized as such if the game visuals do not appeal to the individual playing the game. Because graphics and sound implementation can be time consuming and - depending on the developer's artistic potential - costly, We find it fascinating to investigate the complex balance between logistic functionality and visual elements, which is sometimes difficult.

We divided this implementation category into two main subcategories based on these motivations:

- **Graphics** – All code directly related to the game's graphical appearance is represented here.
- **Sound –** Represents all sound-related code in the game.

A.2 Prototyping and Measurement of Complexity

The thesis method's second main procedure is purely practical in nature. The actual implementation of the Implementation categories into a platform game prototype is included in this procedure. It also includes all iteratively performed complexity measurements on the prototype.

B. Simple Comparison Table:

Game	Created By	First Edition	Software Used	Platforms	Online/ Offline	Genre	Features
Super Mario Bros	Nintendo	1985	Written Directly in 8/16 bit machine code	Nintendo, Wii, Gameboy, ios, Android	Both	Platformer, Adventure	Multiplayer, arcade platformer
Sonic the hedgehog	Sega	1991	Hedgehog Engine	Gameboy, Wii, Nintendo, ios , Android	Offline	Platform, Action, fighting	Speed based Platform gaming

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 7.39 ⁜ Vol. 11, Issue 4, April 2022

DOI: 10.17148/IJARCCE.2022.114135

Limbo	Microsoft Game studios	2010	Custom Engine. Visual Studio	ios, Android, Nintendo, PS	Offline	Puzzle, Platform	Aesthetically Beautiful
Space Platformer	Manas, Muskan, Nishit, Devansh	2021-2022	Unity Engine	Windows, ios, Web	Online	Platformer, Competitive, Puzzle	Better Game Mechanics, Smoother Gameplay, Online Leaderboard

RESULT AND CONVERSATION

A. Implementation –

Platforms, collisions, and the fundamentals of mechanics The first prototype implementation resulted in three subcategories in the Implementation categories being functional. Platforms, collision, and basic mechanics were the subcategories.

Additionally, one property from the subcategory obstacles is included in this implementation in the form of deadly gaps between platforms. This is because deadly areas are essentially just platforms that cause death when they collide.

The level of complexity associated with a platform was another factor that had to be considered. Due to extensive collision, the use of diagonal platforms will increase software complexity, and the use of mobile platforms will generate additional code in the form of platform mechanics. Due to the time constraints of this study, this thesis will present a moving platform implementation that excludes diagonal platforms. The decision was made on the basis of the assumption that moving platforms are more desirable than diagonal platforms.

The first measurable implementation resulted in the following classes and their basic functionality, as shown in Figure:

- **Program -** the application's default static XNA class for initializing and running it. Controller, Input Handler, Game Controller, Game View, and Game Model are all initialized by the Master Controller class.
- **GameController -** Updates the Game Model and Game View, as well as requesting any user inputs in the form of avatar mechanics such as movement and jumping from the Game View.
- **View.GameView -** The main view-class is responsible for initialising and updating the Input Handler and Camera classes, as well as rendering the level and its elements.
- **View.InputHandler -** Checks for new user inputs on the keyboard, gamepad, and mouse.
- **View.Camera -** A class for controlling the avatar's camera and following it around the level.
- **Model.GameModel -** The main model class, which is responsible for initialising and updating the TMX Level and Player Logic classes.
- **Model.TMXLevel –** It is a class that handles all TMX data loading and handling (map file data). Collision handling for all associated map objects is included in this class. The Moving Platform class is also initialised and updated by this class**.**
- **Model.MovingPlatform -** A class for updating a mobile platform's movement mechanics.
- **Model.PlayerLogic -** Acts as a bridge between the Player and TMX Level classes, allowing the avatars to collide and position themselves in relation to the map objects.
- **Model.Player -** It is an object class that derives from Unit and represents the player and its properties.
- **Model.Unit –** It is a class that any Unit object with similar properties, such as players and enemies, can inherit.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 7.39 ⁜ Vol. 11, Issue 4, April 2022

DOI: 10.17148/IJARCCE.2022.114135

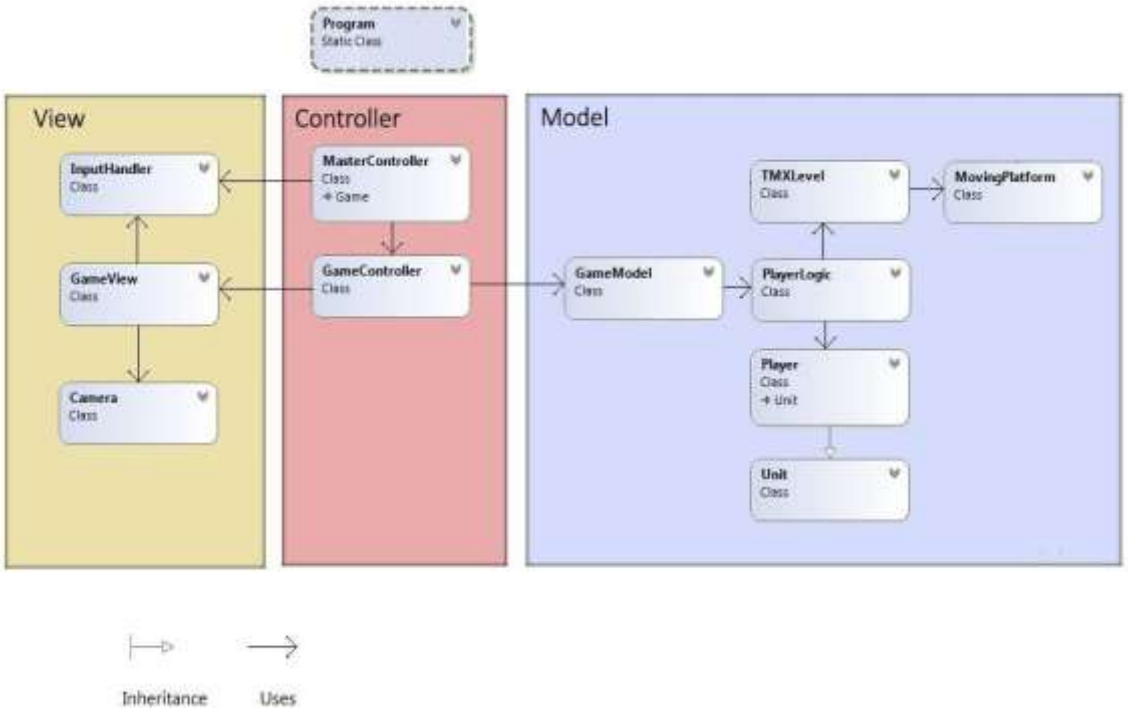

Figure Class diagram showing the basic class relations from the first measurable implementation

It is critical to emphasize that the reason for this first implementation's relatively large class size is that the software's basic structure is included. TMX Level, Moving Platforms, Player Logic, Player, and Unit are the classes directly linked to the actual sub categories platforms, collision, and basic mechanics. We'll treat the view classes and the Game and Master-controller as game structural components for clarity's sake, allowing us to focus on the model-classes that actually implement the subcategories.

Output

Because the Level Visuals category was not addressed in this implementation, the output simply shows a level with platforms and a rectangular avatar. The diagram depicts the result of the initial implementation After that, the arrows were added to help clarify the functional mechanics.

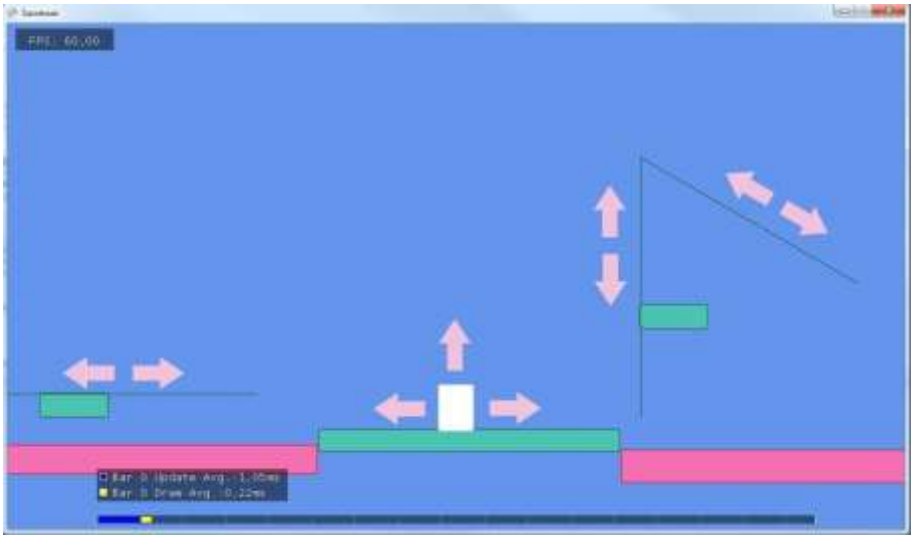

Figure Screen dump from the first implementation's output

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 7.39 ⁜ Vol. 11, Issue 4, April 2022

DOI: 10.17148/IJARCCE.2022.114135

III. CONCLUSION

Despite a handful of concerns regarding the thesis validity we proclaim this thesis to be successful in regard of its original goals. We believe that the thesis results can be of assistance to any small-scale project and that its method could be applied within software engineering in general.

IV. REFERENCES

[1] G. Smith, M. Cha & J. Whitehead, A Framework for Analysis of 2D Platformer Levels, Conference Paper, UC Santa Cruz, 2008
[2] D. Boutros, A Detailed Cross-Examination of Yesterday and Today's BestSelling Platform Games, 2006 [3] E. Zimmerman, K. Salen, Rules of play: Game Design Fundamentals, MIT Press, 2003
[4] A. Ernest, A. Rollings, Fundamentals of Game Design, Chapter 12, Paerson Education, Inc, 2007
[5] J. Ernest, Fundamentals of Game Design, 3rd edition, Chapter 4, New Riders, 2007
[6] C. Pedersen et al. Optimization of Platform Game Levels for Player Experience, University of Copenhagen, 2009
[7] J.P Zagal, M. Mateas, Towards an Ontological Language for Game Analysis, College of Computing and School of Literature, Communication and Culture Georgia Institute of Technology Atlanta USA, 2005
[8] S. Björk, J.Holopainen. Patterns in Game Design, Charles River Media, 2004
[9] K. Compton, M. Mateas, Procedural Level Design for Platform Games, Literature, Communication & Culture and College of Computing, Georgia Institute of Technology, 2006
[10] L.H Rosenberg et al, Software Quality Metrics for Object-Oriented Environment, Crosstalk Journal, 1997