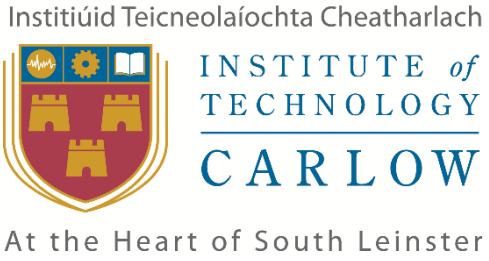


Computer Games Development
Project Report
Year IV

Michal Krzyzanowski
C00240696
28/04/2022

Contents

Acknowledgements

I would like to thank my project supervisor, Martin Harrigan of IT Carlow for supervising this project and helping me greatly with ideas and directions.

Project Abstract

This research is focused on the implementation of a physics-based level editor that estimates the difficulty of a user created level without the use of AI. The levels created are based on the popular game, Angry Birds. The level designer places different shapes around the game scene including the targets & the player position. When the player plays the level, he/she has a limited amount of shots to make (three in this case) in order to destroy all of the targets in the level. The player can also regulate the power of each shot. If all of the targets are destroyed, the player clears the level. In order to get an estimation of the level's difficulty rating, The AI must simulate how the player would play the game and make the shots. AI based level difficulty estimation provides excellent results but tends to be complicated implementation wise and takes a long time to produce the results. My approach aims to provide fast result feedback and attempts to simplify the implementation.

In this paper, I show that it is indeed possible to attain such an implementation albeit losing out on the accuracy of the AI based implementation. The approach taken was to run through all of the possible values when taking a shot from the two variables: **Shot Power** & **Shot Direction**. Each shot would go through a degree-based direction (e.g. 45°) and then select a different power increment. Each shot would be given a **Shot Score** to deduce how good/bad the shot was. Each shot of score greater than 15 is also marked. Once all of the choices are exhausted, The shot with the highest score value is selected and the state of the level of that shot is carried over to the next bullet. This process is repeated for all three bullets or if there are no targets remaining on the level. To further increase the speed of the simulation with a small impact on accuracy, I decided to only use two power ranges (lowest & highest) and I incremented my direction by 10° every angle iteration e.g. 10°, 20°, 30°. The difficulty is then estimated based on the amount of shots that had greater than a score value of 15. Based on the results I have gotten from running this simulation on both a trivial & non-trivial level, the system works as intended while being fast & reliable.

The main difference of my approach to other approaches that I have come up in my time researching this area is that I do not use AI whatsoever. I tackled this problem to the best of my abilities with a very simple & fast approach.

The remainder of this paper is structured as follows: Section 1 is covering the introduction to the project & outlines the research question. The following section, section 2 provides my research into two separate literatures, similar to my project. The third section covers the results I have gained when testing the simulation. In section 4, I provide my project milestones & a review of the project, alongside a conclusion of my research on this project. In the following section, the final section, I discuss any tips for anybody planning on working

in a similar area in the future & I provide links to any of the references that I used in my research.

Project Introduction and/or Research Question

I am creating a physics simulation software using box2D and SDL2. Box2D is an open source 2D physics engine that allows the user to simulate real world physical systems such as rigid body dynamics. SDL2 (Simple DirectMedia Layer) is a library that allows the user to draw images and other primitive shapes to the screen, in addition to the usage of audio, and keyboard, mouse, and joystick events. My physics simulation software will give the user the ability to create savable and loadable simulation levels. The user will be able to place shapes onto the screen, once the player plays the simulation, box2D's physics engine will simulate how the shapes placed would behave in a physical environment. The simulation will come with a predefined list of primitive shapes (rectangle, circle, etc). Any of the primitive shapes can be edited by adding new vertices, changing the position of existing vertices, or deleting existing vertices. Editing shapes will change how they behave during the simulation. This software will also feature joints, an ability to connect different shapes together.

I have decided to work on this because I wanted to learn how to use box2D for possible future game projects of my own and to get a deeper understanding of simulating physical environments.

I will be using this physics simulation to create a game similar to Angry birds, where the player is tasked with hurling a projectile at a structure created with different physics-based objects. The structure would house targets which the player must hit in order to clear the level. My physics simulation, which I explained in the first paragraph of this section, will be used in the game's level editor. The level editor will feature a difficulty estimation program that will attempt to deduce the difficulty of the level using the game's variables: shot power & shot direction, without any AI usage, which brings me to the research question:

Research Question

"Is it possible to create an intelligent level editor that deduces the difficulty of the level being created at runtime without the use of AI?"

This level editor would achieve this by checking how many moves the player has to make to clear the level. The accuracy of the player's shots would be taken into account also. Each shot would be given a score based on how good/bad that shot was. Using the scores, the difficulty estimation software will find the best scoring shot after a full bullet iteration is done and pick the level state for the next bullet iteration.

Literature Review

Exploring Game Space Using Survival Analysis [1]

This paper, written by Aaron Isaksen, is studying the difficulty of a game's generated level by using AI. I do not plan on using AI in my project but, this is the most similar topic I could find to my proposed "Intelligent Level Editor". The paper uses Flappy Bird, a simple score-based game where the player controls a bird that must navigate through a series of pipes without crashing. In summary, the process of analyzing the difficulty is as follows:

1. setup game parameters such as gravity.
2. generate the flappy bird level.
3. note the score(how far the AI travelled before failing) of each AI on a graph.
4. repeat steps 2 & 3 as many times as necessary (20 times in this paper).
5. Analyze the scores of all the simulations.
6. Output the difficulty based on the analyzed scores.
7. Explore game parameters.

The game parameters are mentioned to be: pipe separation, pipe width, pipe gap, pipe gap location range, world height, jump velocity, bird velocity, gravity, & bird size. Each of these parameters would be altered to study the change in difficulty. The level is then generated based on these parameterizations.

The level is then simulated using a player model to play through the level multiple times, each simulation recording the score. The player model is a model the AI will use in the simulation. This model tries to replicate a novice or expert player playing the game. The simulation does one run of a perfect player (makes no mistakes) in order to check if the level is possible to beat. Each simulation also had a max possible score, which made sure that easy games won't play out for too long.

The scores of each simulation are then compared and the difficulty is calculated. Difficulty ranges from 0-1, 0 being trivial and 1 being impossible. The difficulty that was output is then used to explore new game parameters, for example, it could optimize the game parameters for a specific difficulty.

Even though the paper focuses on AI, the goal is very similar to my project's goal, deducing the difficulty of a level created by another user. This paper has influenced me to also store data of the difficulty-deducing simulations, allowing the user who is creating a level to not only get a difficulty rating, but a statistics graph displaying the amount of turns it took the

simulation to clear the level, where each shot was aimed at, & the power used in the shot. The step-by-step approach used in the paper will be used as a guideline when implementing the difficulty-deducing simulation of my own.

Simulating Human Game Play for Level Difficulty Estimation with Convolutional Neural Networks [2]

This paper, written by Phillip Eisen, tackles a similar task as the above paper and my project. The paper is aiming to estimate the difficulty of an ingame level using an AI, which is trained from data gained from actual players of the game. The game in question is "Candy Crush Saga". The paper itself is much larger and more in-depth than the above paper thus, I will only summarize the most relevant sections that I could find.

The process of estimating the difficulty is as follows:

1. CNN (Convolutional Neural Network) training.
2. Gameplay simulation.
3. Creation and evaluation of predictions.

Training the CNN begins with basing the agents around player data recorded from a small subset of actual players playing Candy Crush Saga. The players are also grouped based on their average success rate, essentially allowing the CNN to be based on a variety of skill levels. The player data is separated into two data sections:

- Unfiltered Data: data gathered from random players.
- Filtered Data: data gathered from players who require the least amount of attempts to clear a level.

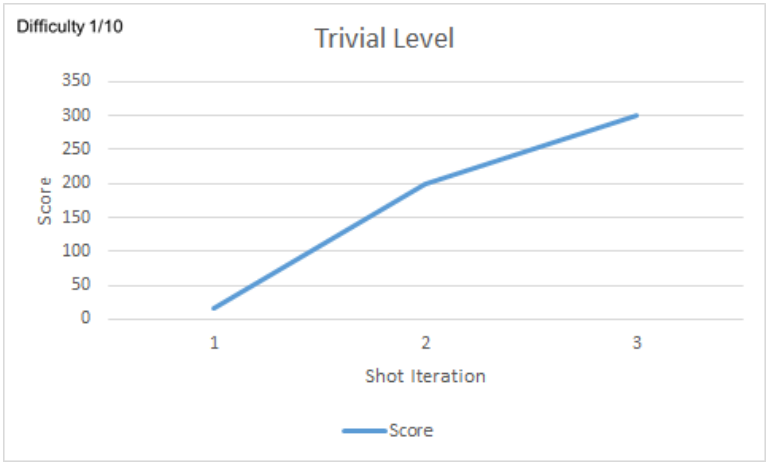
For the gameplay simulation, an interface with an AI agent is created and integrated into the game's engine. The agent then receives possible moves and the current game state, from which it selects an appropriate move. The agents are run multiple times per each level, with some constraints (time constraints/limited amount of moves).

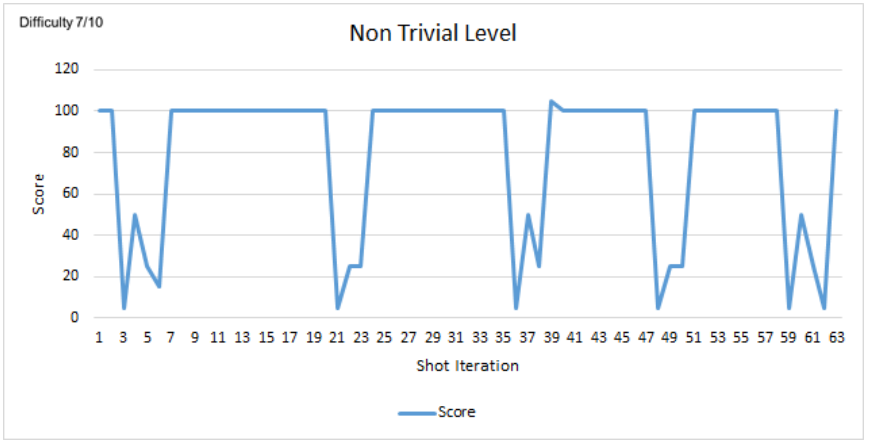
The SR (Success Rate) of the data gathered from the players is then queried and the further player SR is predicted. The predicted player SR is then evaluated by calculating the mean absolute error & root-mean-squared error.

This paper, although focusing on simulating player choices using AI agents, has a very similar workflow to the previous paper.

Evaluation and Discussion

After I implemented the difficulty estimation program, I created two levels for gathering some data, a trivial level that consisted of 3 targets grouped together on blocks on the left hand side of the screen, a non-trivial level, that consisted of 3 targets, one in the center of the screen and the remaining two in the two corners of the screen at the ground. The two corner targets were protected by structures made of standard blocks. Here are the results I have gathered:





Shot Iteration: This is the current iteration of the shot.

Score: This is the score earned in that shot iteration.

As you can see, the trivial level only took a few shot iterations and the final shot iteration counted a score of over 300, meaning that all targets were destroyed with a single shot. Meanwhile, the non-trivial level took over 60 shots to fully access and, not a single of these shot iterations reached over 300 score meaning that each bullet only destroyed 1 target at a time.

Based on the results I've gathered, I came to the conclusion that grouping targets together on one side of the screen tends to make a level much easier than properly spacing the targets across the game screen.

Project Milestones

Basic Level capable of placing square and rectangle shapes into the simulation. the simulation should be pausable.

Deadline: end of week beginning 29th of November.

Basic level difficulty estimation functionality showcasing goal of project

Deadline: end of week beginning 17th of January.

Final implementation of difficulty estimation functionality and general cleanup.

Deadline: end of week beginning 18th of April.

Throughout the course of this project, I was focused on implementing one thing at a time every week, with a few exceptions. Generally speaking however, the project was moving at a steady & manageable pace and the project milestones were met often.

Major Technical Achievements

Couple of things come to mind. I think that setting up Box2D user data to be used in both the rendering and collision callbacks was a big thing to achieve for me. I spent nearly a full week trying to figure out how to set up my ConvexShape class's data to be used as user data for the Box2D shapes. It wasn't easy since the official Box2D documentation on user data was sparse and any bit of information on the web was from old Box2D builds. Another technical achievement I believe was the difficulty estimation program. I spent the majority of my time working on it and it turned out to be rather nice, considering I tried to keep it as simple as possible.

Project Review

The project was moving slow and steadily for the most part. Things that went right were the main game setup with Box2D, I had only a few issues working with Box2D when setting up the core game. What I believe went slightly wrong was the difficulty estimation program. It was quite hard to achieve high accuracy in the estimation while making sure the simulation

ran very fast too. I had plans to implement different ways of estimating difficulty such as using mini-max or implementing a method that checks how many shapes block the targets, this would have been a very fast method if implemented. I was also planning to implement shape editing so that the player could edit the preset shapes into anything they wanted to, but that didn't get implemented since I had issues setting up the rendering for the complex shapes. I don't think my technology choices were entirely wrong but, It would have been much easier to work with SFML than SDL2 for rendering since I'm more experienced with it. My advice to anybody starting a similar project would be to take a minute to plan out some of the more complex functionality, it helped me a lot when programming the difficulty estimation program.

Conclusions

At the end of the project, I was happy with the result of my difficulty estimation program even though it was simple & it lacked another method of estimating difficulty for comparison's sake. I have gained wanted experience in box2D & I have successfully improved my understanding of SDL2 as well.

Future Work

Definitely find and read up on any similar projects being done. It is quite hard to find a project like this done without the use of any forms of AI so, reading up on AI-based versions is good enough, as long as the end goal of the project is similar.

References

[1] Exploring Game Space Using Survival Analysis

[2] Simulating Human Game Play for Level Difficulty Estimation with Convolutional Neural Networks

Paper on optimal convex shape decomposition

Book on computational geometry