Theory of Automata

Dr. Fisher

Hunter Mast

4th December 2023

Project Overview

For my project, I decided to create my own language called “SmashScript”. The language is based on the video game “Super Smash Bros. Melee”. Released in 2001 on the Nintendo GameCube, this game has been a staple of my childhood and friendships. It is one of the most popular video games still today with a very active group of players still competing in tournaments worldwide. Knowing this, I decided to try and create a language and grammar for such a game to see if it would be possible to simulate a match.

When I started this project, I had the idea of creating a compiler. When I had discussed it in class, you had brought up the idea of creating a simple language for it. The idea for that actually never occurred to me before then, so I decided to go with creating my SmashScript language. For the language, I worked on a tutorial to start learning the basics. I didn’t want to just follow a tutorial for the project, so I used it just for a baseline and to get an idea of how languages are created. It is included in the “Testing” folder if you want to see the link to the tutorial or what it was I created with it. For the actual project, I started out with creating a lexer.

A lexer, from my understanding, is where the terms are defined in the grammar. With it, we create token types for every possible word or expression that could be given to us. A token is every individual character, but with how I used it, it should be every phrase in the input. I used RegEx to parse these tokens so I did not have to calculate for every character and keep track of the position of the beginning and end of the string. This is only because I do not have a lot of terms in my grammar and not a lot that are the same. If I was creating a large, fully fledge grammar, then I would look through each character specifically.

After we have our tokens taken apart, we send it to our parser. The parser goes through and defines our grammar. At the beginning of this project, I thought the grammar was defined in another file and then called over to this one. Yet, I was wrong because the grammar is actually coded in how we parse the strings. It defines what we do with our tokens, or another way to say it, what output we give for our input. We start by parsing what “Player” we have. It makes sure we are talking about the “Player” first, then gets the next integer and that is our player. Next, we check if we are performing a move. This line is not necessarily needed, but I did want to include it just in case I wanted to expand this out more to where I can add a “define” action and say what each move is or damage it will output.

Inside of my parser class, I also created a function for specifically going through each of our moves. This function Just will check with the tokens to see which move we perform. There is “attack”, “special”, “smash”, “tilt”, and “grab”. With these, “special”, “smash”, and “tilt” all have variants of the moves that include “up”, “down”, “left”, and “right” moves. “special” even has another variant in the “neutral” move. Each of these do different damage and are different moves technically in the game, but performed in similar ways. I wanted to try my best to include these in some way. Having them as separate moves was the easiest way. The “grab” move is the weirdest move here. You can perform a grab and throw a character, doing damage. You can also grab them, attack, then throw. This was the most complicated to implement because you have attack already as a move, so I have to include it as you perform attack damage then also throw damage. This move seems to be a bit buggy and not work exactly as intended, but the rest work perfectly.

We have an a function to calculate damage and make sure that the correct damage is done depending on the move, character, and player. This then ends us off with a way to check if we win or not. In the actual game, you have a percentage as your health rather than an actual health bar. When you have a higher percentage, you get knocked back farther. Falling off the edge is how you lose a life. The way I decided to simulate this is if you are over 150%, which is what I remember typically when players lose their life, you have a 50% chance of winning of losing. It randomizes your chance and the game either ends or continues depending on that.

A function that is separate from all of this that I do not believe I went over that is included in this is the character select file. This goes through our list of characters and makes sure that you select a viable character. It was easier for me to just include this in another file and call it over. I could easily add more characters such as Luigi, Princess Peach, Ness, Pikachu, and more using this function. It is called into the main one where it is use to see damage and moves that each of these characters can perform.

Super Smash Bros. Melee has 25 characters, which is a lot. Some of them have different attacks and moves. Fox, one of the characters I included, has a different attack as his can infinitely hit compared to Mario, who’s attack can only hit 3 times. This is something I did not really implement, but could easily modify to account for. I just wanted to show a baseline of the grammar to show how all of this could potentially be added for a full SmashScript. Also, the characters lose by going off the edge, so including some movement and figuring out where each player is on a board would be cool. Could change how winning and loosing is done then as instead of a 50/50 chance, it would knock you back and if you are out of bounds, then you lose.

Overall, I learned a lot from this project. Beforehand, I knew absolutely nothing on this. I did know theoretically how this would work with checking the grammar, but not in actual code. Seeing this in action is very cool and something that I may fully realize if I am able to find the time. The biggest problem I kept running into was parsing the tokens into the grammar. It would not register how Player 1 and Player 2 is and how the whitespaces worked. I think it had to do with he “tokenize” function not properly dealing with these whitespaces. Something else I would love to try and work on with this is create a compiler for this. Compiling all of this on it’s own would feel more like an actual game and I believe would potentially create more customizability with it. In conclusion, SmashScript is a language and grammar focusing on simulating a match of the game, Super Smash Bros. Melee. It uses a lexer and parser to see the accepted tokens and output different damage to each player character hit. This project was a great project and something that I hope I can improve on and fully realize someday.

Resource Used for Learning:

<https://www.youtube.com/watch?v=Eythq9848Fg>

Resource Used for Character Information:

https://supersmashbros.fandom.com/wiki/Super\_Smash\_Bros.\_Melee