Slippi Stats: Final Planning

University of Wyoming
Senior Design 2023-2024

Beckham Carver bcarver4@uwyo.edu, Kyle Lofthus klofthus@uwyo.edu,
Zachary Crimmel zcrimmel@uwyo.edu, Michael Stoll mstoll3@uwyo.edu Ben Wilkin
bwilkin@uwyo.edu,

(1) An overview/executive summary of your project

The goal of the project is to create a highly modifiable/tunable structure for making bots for every character in Super Smash Bros Melee (SSBM). With an easy to use UI for tuning bots for novices to programming, with a scalable backend. SSBM is an extremely dynamic game with exceptions to every rule it gives. The project will implement libmelee which is a library in Python that interfaces with Slippi to create virtual controllers for bots. It contains additional tools for parsing gamestate, handling menus, looking up frame data, and gives useful enumerations for some of the weird numeric assignments SSBM uses at the hardware level.

The BotBuilder will need to have a generic structure to account for the Melee'isms (as they're colloquially known) of SSBM. We plan to do this by setting up a generalized layer of attacks/actions (GeneralizedChains) that can then be automatically parsed on a per-character basis for the best fitting move to that action. To explain the importance of generalized chains, SSBM has roughly 5 types of attacks, each of which has a unique version depending on the direction you hold the analog stick when using them. These are Tilts, Smash Attacks, Aerials, Specials, Grabs/Throws, and some other oddballs.

- Generally, each type of attack serves a specific purpose, smash attacks hit hard and are slow, tilts are for quick zoning, aerials combo, and grabs are for punishing bad movement.
- Generally the direction you hold the attack will cause the opponent to be hit in that direction, or extend your character's hitbox in that direction.
- But not every attack abides by these rules,and practically every character has at least 1 or 2 moves that break the convention.
- You cannot rely on down + aerial moving an opponent downard, or up + tilt to launch an opponent above you. There will be more than one exception in the 26 character cast for both cases.
- To program a bot that can play every character optimally, you would have to completely rewrite the logic for move choice for each character to account for this variance.

The goal is to have the BotBuilder be able to play (or make bots for) any character, so it will use GeneralizedChains (elaborated later) and an algorithm with predetermined

moveSubsets in CharacterChains to verify the most accurate move for a request. GeneralizedChains will be of the form StrongUpAttack, or DownAeriel, and will then be converted to character specific moves at runtime.

BotBuilder will implement a Strategy > Tactic hierarchy to decide chains, and allow the user to switch the ordering in which tactics should be used to customize bot behavior. Tactics will contain rating functions to check their applicability during gameplay, and choose the appropriate tactic. Tactic rating function weights will also be accessible to users.

(2) The current team (you!) with short 2 sentence bios as well as what "role" / lead you are each taking in the project

- Beckham Carver: Team lead, project designer and will work on backend interfacing and overall code design. Knows the melee metagame well and some of its weird quirks.
- Kyle Lofthus: Will be working on backend, designing the bots and implementing how the Tactic weights will be applied to the bots. Potentially using machine learning to switch between different tactics based on in-game circumstances.
- Michael Stoll: Backend programmer, will work on moveSubset search algorithm. Also potentially interfacing between backend and frontend.
- Zachary Crimmel: Designing layout and functionality of different fields in the application to allow the user to customize the bot. Create locations for the user to save the different bots that they create
- Ben Wilkin: Frontend programmer, working on UI in Python and any graphic design elements. Good knowledge of SSBM game mechanics and meta.

(3) A description and/or list of the functional requirements of your end product and/or minimum viable product (what must the final or MVP be able to do?)

- **Summary:** Generalized bot that can play any of the 26 characters in Super Smash Brothers Melee programmed in Python with LibMelee. BotBuilder will have a frontend for modifying bot parameters in real time and saving them to create new bots.

- **Backend:** Must have a bot that follows are Strategy > Tactic > GeneralizedChain > CharacterChain model
    - Implement Strategies that call basic Tactic.rating functions
    - Implement GeneralizedChains for at least 4 types of attacks
    - Implement a CharacterChain algorithm for choosing the best move from a moveSubset on any character
    - Be able to load Strategy, and CharacterChain from JSON files

- **Frontend:** Have a UI for creating, loading, and modifying existing bot profiles.
    - Load: Loads bot JSON files in visualizes their parameters
    - Edit: Allows user to view the list of available tactics, and rearrange or omit them from a strategy
    - Edit: Allows user to add a CharacterChain to their bot for a given character, and modify its moveSubset [{moveType : buffer }] adding, removing, or rearranging moveSubset.
    - Save: Saves any modified parameters into a bot JSON
    - Play button: Passes loaded JSON data to the backend, and instantiates a bot instance inside of Slippi

(4) A High-Level Systems Overview [e.g. what are the major components in your system]; Diagrams and Text descriptions would be appropriate

Definitions & Preliminary Information:

- Frame disadvantage: the number of frames (SSBM runs at 60fps) that the opponent is stuck completing an action. This is directly accessible from the gamestate.
- Frame data: the length and types of frames of a move, including startup in-active frames, active hitbox frames, invulnerable frames, and ending lag frames.

Backend:

Bot is programmed to make decisions in a hierarchy of strategy > tactics > generalized chains > character chains. This is largely inspired by SmashBot which is a bot that exclusively plays Fox. Though we will follow a similar pattern of delegation of game and player_states, SmashBot is highly specified to work only with Fox, and even with heavy modification would only work with Fox.

- Meta/Agent Level:
    - User can define what character(s) this bot will be able to play. Agent manages menus and other bookkeeping.
- Strategy:
    - Strategy can only be set at the start of a game, and will determine the gameplay loop for the duration of that game.
    - Calls tactic rating functions in a predetermined order, delegating work to the tactic if its rating function accepts.
    - Tactics can (and will) be switched if its rating function is no longer true, as long is it marked interruptible
    - The order in which tactics are called will be modifiable from the UI, and therefore needs to be instantiated at runtime via a JSON file.
    - Uses constructor for mapping the order in which tactic rating functions are called
- Tactic:

    - Tactics are switched dynamically and determine sections of gameplay, largely determined by stage position.
    - Contains a rating function that determines if the given tactic is viable, variables used in the rating function will be modifiable from the UI.
    - Calls generalized chains that accomplish subgoals of the tactic, such as ToPlatform followed by UpAttack in a loop for a vertical juggle tactic.
- Generalized Chain:
    - Uses generic language for the types of attacks
    - Interface for Tactics to be able to call the correct CharacterChain, and have language for defining gameplay decisions across all characters.
- Character Chain:
    - Searches through a subset of the possible moves (user definable) for the best action.
        - Each move in the subset contains a user definable frame buffer, which is added to the opponents frame disadvantage. This allows the user to set the appropriate risk worth taking on any given move.
        - Prioritizing the order of the user defined list. Checks each move's frame data against the opponent frame disadvantage + buffer, checking if players current trajectories collide during active frames of the attack. If an attack can hit the opponent during their frame disadvantage it is used.
        - If no attack can currently reach the opponent but the opponents frame disadvantage is greater than 0, set the chain to be interruptible and move towards the opponent. Otherwise do nothing.
    - The above model works for most attacking & defending chains. However for movement based chains more granular attention will need to be given as the 26 character cast of melee has varying types of optimal movement.
        - Generally there are three types of movement: grounded, aerial, and wavedashing.

        - A preliminary approach may be to allow the user to set the preferred order of these movement types, and let character chain class use the preferred method whenever possible.
        - In the early stages of development it may be easiest to assume grounded movement is optimal on all characters.

Frontend <-> Backend Interop:

- Because the BotBuilder will need to be able to modify core variables in the backend from the frontend, a robust system of communication between the two will be needed.
- Additionally, as new Tactics and Chains are made, front end designers will need to be able to easily integrate their variables into the UI. Front end designers may not always know the true purpose of a variable in any given Tactic or Chain on first pass.
    - Backend programmers should devise a system of commenting that allows the python source files to be parsed as text, and search for variable definitions and names. This way if a variable's purpose is changed, the frontend designers do not need to write new descriptions and they can be dynamically loaded.

Frontend:

- Creates a UI that allows the users to toggle values that determine how the bot prioritizes "Strategies" and implements advanced selection options for setting the rating function for the "Tactics," and basic logic for "Character Chains"
    - Home Page: Can select or create a 'bot' which starts with default parameters.
        - (Stretch) can select between different preset starting params chosen by the team

- (selected bot) Meta page: allows user to set meta details, such as stages the bot is willing to play, or the character(s) it will play.
    - Called from the agent, menu related actions only.
    - (stretch) Add logic for selecting different strategies during menu
- (selected bot) Strategy page: Can modify the strategy(s) for the bot
    - Modify: (rearrange or omit) the order of a predetermined list of tactics in the if/else chain found in strategy.py
        - Users are shown a list of tactics, can drag them up and down, or unmark them to omit them.
    - (Advanced & stretch goal) create: can create additional strategy.py files for the same bot, logic for selecting a strategy can only be called in the menu state, as a new agent will need to be created. *Strategies are not meant to switch during gameplay, this is the job of tactics.*
- (selected bot) Tactic Page: Can modify the rating values/functions from a predetermined list of tactics:
    - Users are shown a list of selectable tactics, those omitted in strategy are grayed out
    - On selected tactic, user is shown its list of accessible variables and a summary of the function they are used in
    - Modify: ie. change “if x.distance < 20 return forwardAttackChain” to “if x.distance < 30 return forwardAttackChain”
    - (Advanced & stretch goal) can open submenu to add additional if statements to a tactic, using a stack based statement builder.
- (selected bot) Chain Page: Can modify the list of valid moves in any of the characerChains:
    - Modify: (rearrange, omit, or add) from the list of all possible moves in melee (tilts, aerials, smash, etc.)
    - CharacterChains should have a default moveSubset for all unimplemented characters by the user.

- Users can save the ‘bot’ as a JSON containing the strategy values/order, tactic rating function weights, and character chain moveSubsets.

(5) List and Detailed Description of the MAJOR Milestones for your project (think top 3-4 things) -- How will you know when you have successfully reached/completed each milestone? && **(6)** Major tasks that lead to each milestone.

- Design a character non-specific bot that follows a basic gameplay loop. Uses a single strategy and three tactics: recovery, approach, and an attack. Each tactic implements the minimum amount of chains needed (ie. jump, forwardAttack, specialAttack, and move). All actions are done irrespective of character, generalized and character chains are identical to this point. ie. Up+B is assumed to move vertically on all characters.
    - We will know we have reached this goal when we can play against the bot and clearly observe its small gameplay loop.
    - Removal of tactics should limit its behavior.

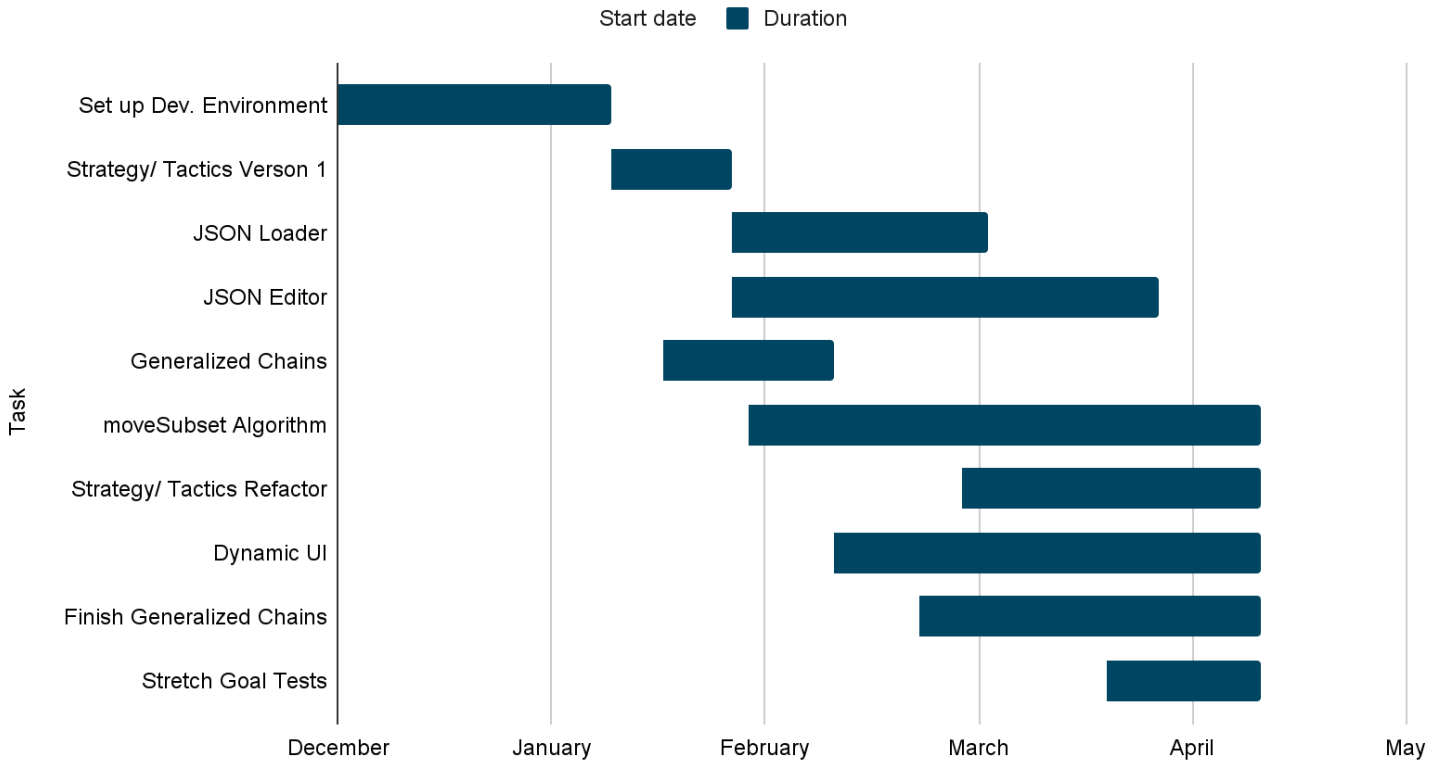
- Create UI features to launch Slippi with the bot connected, and load in moveSubsets from the file system. Implement characterChains, and a basic search function for evaluating if moves can hit dynamically using frame data.
    - We will know we have reached this goal when we can observe the bot only using moves listed in its moveSubset.
    - Changes in the frame buffers should make the bot more and less reckless.
    - The bot should be able to accurately predict future position based on both players' velocity.

- Implement robust Strategy and Tactic classes, reorder tactics based on botJSON, and make tactic rating functions easy to modify. Create UI features to generate and load botJSON files, being able to modify strategy as described in the systems overview. Pass along the botJSON file to CharacterChains. Interoperability between the front and backend will need to be assessed and the UI will need to be as dynamic as possible.

- We will know we have reached this goal when we can observe the bots priorities changing as tactics are rearranged in the UI.
- Basic modification of tactic rating functions should change bot decisions
- Addition of new Tactic files should create new UI elements dynamically

- Program GeneralizedChains for all basic attacks and movement options that have not been made yet. Testing their translation into CharacterChains and modifying the CharacterChain search function as new bugs are encountered. CharacterChains should be able to switch between aerial and grounded attacks dynamically based off of the moveSubset. There should be GeneralizedChains for all types of moves.
  - ‘Fast’, ‘Strong’ and ‘AerialOnly’ versions of: ForwardAttack, ReverseAttack, DownAttack, UpAttack, ZoneAttack, and ProjectileAttack
  - Special chains for: TechChase, JabReset, Grab, Throw, and Rest
  - Movement chains for: MaxLeft, MaxRight, MaxVertical, ToLedge, and ToPlatform.

Slippi Project Timeline



(7) A list of any additional/stretch goals that could be added to the project if time allows.

- Addition, setup agent instantiation to allow bots to play against each other. This should be a simple feature to add, but is not required at any point in the core development.
- Stretch goal, implement a way to read in a set of .slp files and use the generated stats to output a basic set of values/orders for a bot. This would not involve any machine learning, but be a direct function. Could interface with the Slippi Database.
- Stretch goal, utilize the bot builder parameters as a genomes for a genetic algorithm. According to Vlad Firoiu et al. they were able to emulate 50 instances of SSBM at ~1-2x speed on 50 CPU's for RL and Q-learning. Given similar throughput at 1.6x speed and a maximum gametime of 8:00 minutes. We could emulate a rotating population of 100 at 6 generations per hour. This may
- Stretch goal (alternative to above), feed in .slp gameplay as a replay and compare live gameplay to bot decisions. Could be used as a feedback system, or in a prediction model where tactics and variables are modified. May be more feasible to run on consumer hardware as replaying files is faster (and easier to tune) than live gameplay.