# SMES: Adapting Dexterity-based Games for Deliberative Play

**Batu Aytemiz, Hunter Lynch, Carl Erez, Adam M. Smith**

University of California, Santa Cruz
1156 High Street
Santa Cruz, California 95064
{baytemiz, hflynch, cerez, amsmith}@ucsc.edu

**Abstract**

In this paper we describe the Sharp Multi-input Editing Software (SMES), a prototype system which allows players to convert dexterity challenges to planning challenges in existing games. SMES accomplishes this by recording the player input, displaying it in a visual form, allowing the player to edit the recorded playtrace, and replay the edited sequence back again. Our system combines ideas from tool-assisted speedruns (TAS) with interaction design patterns from Casual Creators and Accessible Player Experiences. Because our implementation is based on intercepting input events at the operating system level, no engineering effort is needed to apply the tool to new games. The result is a new mode of play for existing games that further allows players to realize new behaviors in the game. SMES could support making games more inclusive by letting players adapt the challenge type they face. Furthermore we claim that AI systems that operate over playtraces to assist the player is a rich avenue to explore.

Challenge is a crucial component of videogames (Adams 2014). From solving puzzles to defeating enemies, from scaling cliffs to uncovering mysteries, games often ask their players to overcome a variety of obstacles. The presence of at least some challenges has shown to be important for player engagement and fun (Juul 2009). Yet, not everyone wants to engage with the same types of challenges (Hamari and Tuunanen 2014). One group of players might find dexterity based challenges more fun whereas others might prefer a slower-paced logical puzzle. Furthermore, an individual player might want to engage with different types of challenges at different times.

Beyond personal preference, however, some challenges can be much less accessible to certain groups of players. If one has a motor disability then it is much more difficult for them to engage with a high-intensity dexterity based game (Pitaru 2008). It is one thing for a player to choose to not play a certain type, and another for the player to not be able to play the game.

When faced with barriers to engagement it is common for people to modify games through using the different interfaces to play the game such as switch controls (Sethfors 2018) or Microsoft Universal Controller (Englard 2018), or by changes in software via trainers.

Inspired by the spirit of modifying how people engage with games, in this paper we discuss our work-in-progress *Sharp Multi-input Editing Software* (SMES). The SMES allows converting high-paced dexterity based challenges to deliberative planning challenges in a narrow set of games that exist in the wild. SMES was built with difficult platformers with very short levels in mind such as Celeste (Matt Makes Games 2018) and Super Meat Boy (Team Meat 2010).

SMES allows the players to save, display, modify and replay their keypress sequences. SMES works through the following four steps:

1. The player attempts the challenge and SMES records their keypresses.
2. The SMES visualizes the keypresses over time.
3. The player makes experimental changes to the keypresses.
4. The SMES replays the edited keypresses (and the player returns to step 2).

This loop allows the player to change how they engage with dexterity based games. The challenge stops being about the precise timing of button presses, but rather becomes more like a puzzle game where the player incrementally refines the play trace until they clear the level.

When we extract and replay the playtrace of the player, we treat the sequence of actions as a stateless, fixed policy. Similar stateless action sequences have been used successfully for difficult exploration problems (Ecoffet et al. 2019) (Zhan, Aytemiz, and Smith 2018) and is shown to be a promising starting point.

We believe that AI techniques can be effective in assisting players while operating on playtraces. Representing dexterity based games as playtraces, and then recording how player changes allows us to get another type of data as to how a player might improve their playtraces.

In the rest of this paper we will discuss the technical details and the design decisions of SMES, and how the extracted play traces can be a fruitful avenue for AI research.

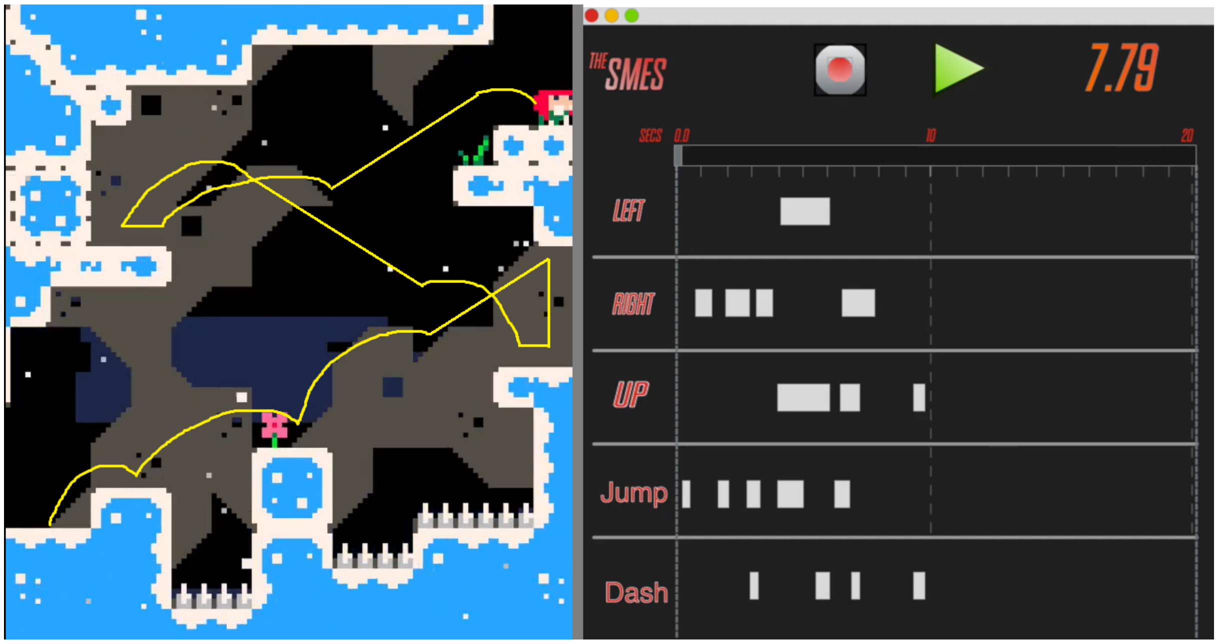

Figure 1: On the left, the approximate path that player took from the bottom left of the map to the top right of the map in the game Celeste. On the right, the buttons press sequence that the player pressed in order to take this path.

## Related Work

Videogames provide intrinsically valuable experiences to millions of individuals (Juul 2011). Maybe more importantly, games are an important part of how people connect to one another (Organization 2020). While multiplayer games offer moment-to-moment connectivity, the connective nature of games also extends to single player games through creating a set of shared experiences (John 2020) and communities (Gee and Hayes 2012).

It is important for everyone to be able to engage in this shared culture. Yet, through a combination of design decisions of developers and differing abilities of players, a significant group of people are excluded from engaging with games (Taylor 2013). To make games more inclusive, in recent years there have been a number of games that include assist modes such as Celeste (Matt Makes Games 2018) and Super Mario Odyssey (Nintendo 2017). These assist modes allow the player to change the magnitude of difficulty they encounter by tweaking variables such as game speed, player and enemy attributes and other relevant factors. The addition of assist mode has been met with positive feedback from the players (Klepek 2019).

Most assist modes do not modify what type of challenge there is but rather they work by changing the magnitude of the challenge. In this project we wanted to explore whether we could change the type of challenge that the game presented. We use the Taxonomy of Failure (ToF) to categorize the types of challenges (Aytemiz and Smith forthcoming). ToF offers six different categories of failure which map to different types of challenges that exist in a game. For this project the relevant categories are Planning and Execution. Planning refers to the challenge category where the player is deciding on what actions to take. The Execution category on the other hand refers to the challenge of the player to actually take those actions, by pressing the correct buttons in the correct order and timing.

In the context of difficult platformer games, which SMES is targeting, the planning challenge would be deciding what route to take and when to use the relevant abilities such as dash or jump, and the execution challenge would be related to pressing the buttons to activate the relevant abilities at the correct time. Our goal in this project is to replace the challenging execution aspect from playing difficult platformers into a planning one.

Instead of building a new game that has the planning challenge present as a default, in this project we explore a tool that targets already existing culturally relevant games and supports people's ability to engage with them. While there have been many successful games that are specifically designed to target groups with disabilities (Grammenos et al. 2006) we believe that due to the social nature of games it is important to support players in engaging with pre-existing cultural artifacts.

In building the SMES we were inspired by the ideas of transgressive play (Aarseth 2014) and the diversity of ways players customize how they engage with games. One example of this is the WeMod community[1]. WeMod allows players to reconfigure over a thousand single player games through a unified interface, letting them customize the game to their liking, often through changing difficulty and progression variables. As of July 2020 WeMod advertises that

[1]https://www.wemod.com

it has over 8 million users. This shows that a significant population of players are accustomed to modifying their play experience by reaching outside the game.

The Speedrunning community is another group that inspired this project. Speedrunners are interested in completing a game as fast as humanly possible. Tool Assisted Speedruns (TAS), take this concept a step further and use emulators to complete a game as fast as possible using any and all available tools. While most TAS systems run the game in an emulated environment to have frame by frame control, in order to increase generalizability of our system we simply replay keypress events (which is not guaranteed to reproduce gameplay over long periods of time) instead of attempting to emulate the whole game.

The disability community has also been customizing the way they engage with games through a combination of software and hardware (Beeston et al. 2018). The Accessible Player Experiences (Cairns et al. 2019) design patterns are a great resource when it comes to making games more inclusive. These 22 patterns target both the ability of the players to access the game content, and their ability to adjust the challenges present in the game. In the later chapter we will be discussing how SMES fits these design patterns.

## Technical Design of the SMES

### Design Inspirations

Before the development phase of the SMES we reached out to different communities on Reddit to inform our design. We had conversations on the disabledgamers, TAS (tool assisted speed run) and pc gaming subreddits. While few in number, the discussions around how such a tool might be used was immensely valuable in prioritizing certain features over others and understanding potential user needs.

Our design was informed by a public discussion of a system prototype on Reddit's `r/disabledgamers` community and `r/TAS` communities

We incorporated *casual creator* design patterns wherever possible (Compton and Mateas 2015). Casual creator patterns aim to make systems more usable and friendly. While these design patterns are traditionally used in the context of generative systems, we found them useful in our design. Like most casual creators, SMES is a tool that prioritizes approachability and flexibility over control and frame-perfect precision.

Finally SMES also drew inspiration from the Accessible Player Experiences (Cairns et al. 2019) (APX) patterns. APX patterns have two distinct categories: Access patterns support players accessing the game by allowing them to tune the experience to meet their unique needs, and challenge patterns on the other hand allow players to modulate the amount of challenge they want to face.

**Recording Key Presses** One decision we had to make early on was what layer of abstraction to have SMES integrate. While extending an emulator would allow us to be much more precise both in recording and replaying our playtraces, it would also limit the games that the SMES would be compatible with to those that could run on the decided emulator. One of our guiding principles while building SMES was to support access to a broader set of culturally relevant games, and we therefore decided not to limit ourselves with any specific emulator

In order to reach a wider range of games SMES uses the Win32 API[2] to integrate into the keyboard input layer. After the player starts recording, the SMES intercepts and saves the virtual keyboard calls that happen while the game is focused. When the player stops recording, the keypresses are saved into a text file. Having players seed the starting playtrace with their own action connects with the "No blank slate" pattern of casual creators. While the players can start from scratch to create the full playtrace using SMES, having a starting playtrace to edit makes the iterative process much faster.

To support the quickest edit-and-replay interaction loop, it is the player's responsibility to include input sequences that would reset the game to a safe state (e.g. accessing an in-game menu to restart the level from a checkpoint). If there is no such obvious sequence to include in the playtrace that will accomplish this goal, the player can always use manual controls (similar to how they recorded the original playtrace) to reset the game before continuing the SMES loop.

**Representing Key Presses** It is common for speed-runs to be shared through a BK2[3] file. These files are a way to record which combination of keys were pressed down at which frame. Inspired by this, our intermediate representation is in a simple human readable text file. This was decided in part to support the easy sharing of play traces. The easy sharing component is aligned with the "Self promotion" casual creator design pattern that acknowledges the shared and social nature of playing games and generation.

Having a semi-permanent copy of the playtrace that the player can keep editing also aligns with the "Save Early, Save Often" and "Slow it Down" patterns of APX. The player can take as many iterations as they would like to edit the playtrace. This allows the player to engage with the game on their own pace instead of being locked to the speed the game demands. Furthermore, even if they have to close SMES and come back later, the text file will be there for them to pick up where they left of

Because SMES can not know what exact frame each button was pressed, instead of matching keypresses to frames, we record the timing of each press. We record the timing of whenever a key is pressed down and later released. However, instead of recording at what time these actions of key_down and key_up happen, we record how much time elapses in the middle. This makes keeping track of multiple keypresses and the duration of a specific keypress easier. To be more specific, our representation is a series of two data points, the action_type and the modifier. There are three possible actions, key_down, key_up, or delay, and each action has a modifier associated with it. If the action type is key up or down, then the modifier is the name of the key that is pressed or released. Whereas if the action type is delay, the modifier is how many milliseconds have elapsed since the last entry.

[2]https://docs.microsoft.com/en-us/windows/win32/api/

[3]http://tasvideos.org/Bizhawk/BK2Format.html

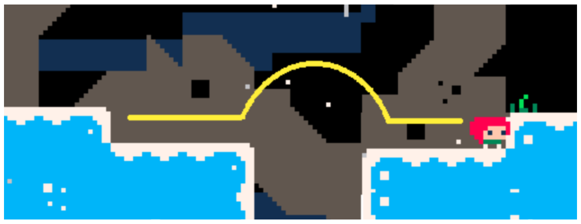
Figure 2: The visual path that is described by the given playtrace.

Let us give an example of a playtrace. Imagine a character starts running to the right, then jumps over a gap, and stops running. This can be shown as follows:

```
key_down , d
delay    , 500
key_down , w
delay    , 1000
key_up   , w
delay    , 500
key_up   , d
```

In this example, the player holds down "d" for two seconds, while pressing "w" for a single second in the middle.

**Visualizing and Editing Keypresses** After the playtrace is saved into a text file, the SMES GUI loads them and visualizes them. The current GUI is written using the QT framework and is still a work in progress. While designing the visualization we were inspired by digital audio workstation (DAW) software packages. SMES allows the users to manipulate the playtraces by changing when and for how long a key is pressed down. When the desired changes are done the updated playtrace is once again saved into the text document.

This design is aligned with the "Improved Precision" and "Personal Interface" patterns of APX. It is an additional interface that the players can use to engage with dexterity games and by removing the temporal aspect of taking actions the player can be more precise. Depending on motor skills and personal preferences, players may have an easier time with our GUI or with a text editing for which they have built deeper fluency (in which case the GUI forms a passive viewer tool).

**Replaying the Playtrace** The player can start the replay whenever they want by pressing the replay button. SMES then starts replaying the playtraces that is saved in the intermediate text file. In order to replay the playtrace we use the underlying Win32 API, the same system we used to record the keys. We use the python package winput[4] to send Virtual Key Codes as input to the operating system directly.

Due to the game not running in an emulator and the Win32 API not being instantaneous the playtrace that is replayed is not completely deterministic. In our experiments there is an 2% average divergence between the length of the input sequence and its realized replay. Because the errors compound in a playtrace this can result in perceivable differences in outcomes across different replays of the same playtrace. This is the main bottleneck that limits the usefulness of the current state of the SMES in supporting longer playtraces or styles of play that rely on precise determinism (e.g. luck manipulation strategies used in TAS). These are not critical limitations, however, because we imagine SMES being applied to localized scenes (which can be re-attempted many times) in games that were otherwise completable within the limits of human dexterity.

[4]https://pypi.org/project/winput/

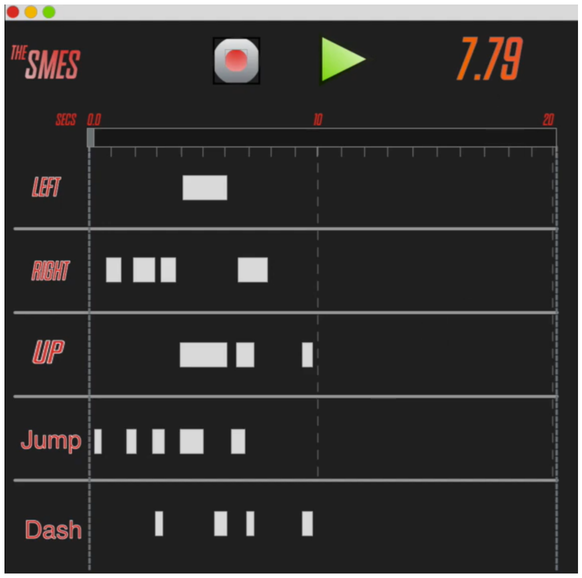

Figure 3: The visual mock-up of the editor. The prototype GUI is actively in development.

## Future work: Manipulating Playtraces with AI

As mentioned earlier, there is currently no AI system acting on the saved playtraces. However, other work shows that by adding a simple level state awareness reusing parts of play traces can be very useful. For an example from the games industry, in the fighting game Killer Instinct, the enemy shadow AI learns how to play by memorizing sequences of player actions (Neal 2016). Then the AI agent replays parts of the player's action sequences when a correct game state is matched. This method proves effective enough to be deployed in a live commercial game, and it has been met with approval from the player base[5].

In the future, we are interested in taking a co-creator approach to supporting playtraces similar to the reinforcement learning based mixed-initiative level creator Morai Maker (Guzdial et al. 2019) or the work on user empowerment where a reinforcement learning agent is trained to augment (rather than replace) player input (Du et al. 2020).

We imagine that the edits that the user makes to the timeline are not directly applied but are instead taken as training data points to tune a stateful and observation-dependent

[5]https://forums.ultra-combo.com/t/shadows-ai-retrospective-discussion/24899

action policy. In this setup, the player is still making deliberate choices about how to act in specific moments, but the local choices they make can have impact on several other moments (e.g. those that arise in the future or as the result of different stochastic elements of the game).

Furthermore, we believe that the datasets describing how a player iteratively refines their playtrace can be interesting for future applications.

## Conclusion

In this paper we presented the work-in progress Sharp Multi-input Editing Tool (SMES) which allows the players of difficult platformer games to change the challenge type from a dexterity based execution challenge to a cerebral planning challenge. SMES does this by recording the actions of a player, and then presents it to them in a visual interface where they can make changes and replay the updated playtraces. We also discussed how playtraces can be operated on via AI systems with the goal of assisting and empowering players. We hope that with SMES we can contribute to the ongoing conversation of how we can use AI techniques to assist our player.

## References

Aarseth, E. 2014. I fought the law: Transgressive play and the implied player. In Segal, N., and Koleva, D., eds., *From Literature to Cultural Literacy*. London: Palgrave Macmillan UK. 180–188.

Adams, E. 2014. *Fundamentals of Game Design*. Pearson Education.

Aytemiz, B., and Smith, A. M. forthcoming. *Proceedings of the 15th International Conference on the Foundations of Digital Games*.

Beeston, J.; Power, C.; Cairns, P.; and Barlet, M. 2018. Accessible player experiences (APX): The players. In *Computers Helping People with Special Needs*, 245–253. Springer International Publishing.

Cairns, P.; Power, C.; Barlet, M.; and Haynes, G. 2019. Future design of accessibility in games: A design vocabulary. *International Journal of Human*.

Compton, K., and Mateas, M. 2015. Casual creators. In *ICCC*, 228–235. axon.cs.byu.edu.

Du, Y.; Tiomkin, S.; Kiciman, E.; Polani, D.; Abbeel, P.; and Dragan, A. 2020. AvE: Assistance via empowerment.

Ecoffet, A.; Huizinga, J.; Lehman, J.; Stanley, K. O.; and Clune, J. 2019. Go-Explore: a new approach for Hard-Exploration problems.

Englard, K. 2018. Microsoft has created the most accessible gaming controller ever made. https://www.vice.com/en_us/article/d3kvx7/microsoft-adaptive-controller. Accessed: 2020-7-24.

Gee, J. P., and Hayes, E. 2012. Nurturing affinity spaces and game-based learning.

Grammenos, D.; Savidis, A.; Georgalis, Y.; and Stephanidis, C. 2006. Access invaders: Developing a universally accessible action game. In *Computers Helping People with Special Needs*, 388–395. Springer Berlin Heidelberg.

Guzdial, M.; Liao, N.; Chen, J.; Chen, S.-Y.; Shah, S.; Shah, V.; Reno, J.; Smith, G.; and Riedl, M. 2019. Friend, collaborator, student, manager: How design of an AI-Driven game level editor affects creators.

Hamari, J., and Tuunanen, J. 2014. Player types: A meta-synthesis.

John, M. 2020. There are no single player video games.

Juul, J. 2009. Fear of failing? the many meanings of difficulty in video games. *The video game theory reader* 2(01):2009.

Juul, J. 2011. *Half-Real: Video Games between Real Rules and Fictional Worlds*. MIT Press.

Klepek, P. 2019. The small but important change 'celeste' made to its celebrated assist mode. https://www.vice.com/en_us/article/43kadm/celeste-assist-mode-change-and-accessibility. Accessed: 2020-5-8.

Matt Makes Games. 2018. Celeste. Windows PC version.

Neal, D. 2016. Designing AI for killer instinct. https://www.youtube.com/watch?v=9yydYjQ1GLg.

Nintendo. 2017. Super mario odyssey. Nintendo Switch.

Organization, W. H. 2020. Games industry unites to promote world health organization messages against COVID-19; launch #playaparttogether campaign.

Pitaru, A. 2008. *E is for everyone: The Case for inclusive game design*. MacArthur Foundation Digital Media and Learning Initiative.

Sethfors, H. 2018. I used a switch control for a day - 24 accessibility. https://www.24a11y.com/2018/i-used-a-switch-control-for-a-day/. Accessed: 2020-7-24.

Taylor, C. 2013. Able gamers includification.

Team Meat. 2010. Super meat boy. Windows PC version.

Zhan, Z.; Aytemiz, B.; and Smith, A. M. 2018. Taking the scenic route: Automatic exploration for videogames. *arXiv preprint arXiv:1812.03125*.