A Demonstration of Mechanic Maker: an AI for Mechanics Co-creation

Vardan Saini and **Matthew Guzdial**
Department of Computing Science
University of Alberta
Edmonton, AB, Canada
{vardan1, guzdial}@ualberta.ca

Abstract

Developing games is challenging as it requires expertise in programming skills and game design. This process is costly and consumes a great deal of time. In this paper we describe a tool we developed to help decrease this burden: Mechanic Maker. Mechanic Maker is designed to allow users to make 2D games without programming. A user interacts with a machine learning AI agent, which learns game mechanics by demonstration.

Introduction

Game development requires specialized knowledge, in terms of designing and coding skills. These skill requirements serve as a barrier that restricts those who might most benefit from the ability to make games. For example, educators, activists, or those who cannot afford the time or monetary cost of learning traditional game development. Researchers have promoted automated game design as a potential solution to this problem, in which computational systems build games without human interaction. However, automated game design doesn't fully reduce the difficulties surrounding game development. These existing systems rely on knowledge about game design inputted into the system by their developers (Cook, Colton, and Gow 2016; Summerville et al. 2018). This means that extending the tool to be able to produce new games requires the same knowledge as if a human was developing the game. In comparison, our tool is made to automatically learn implicit game mechanics from a human user.

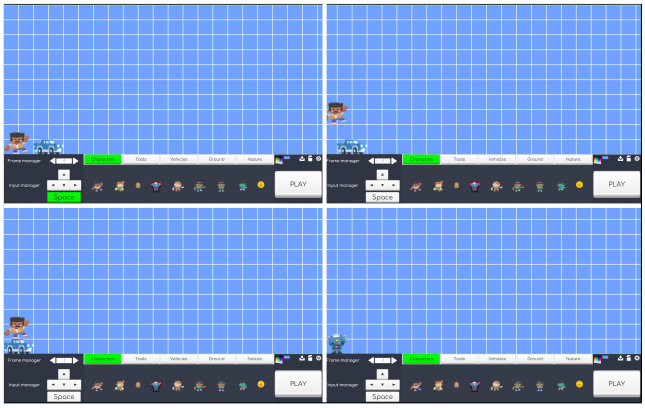
In our tool, a user creates the frames of a non-existent game, demonstrating visually how they would like the game to work. We built out the graphical user interface (GUI) for this tool in Unity, a game engine, to bring user creations closer to end products. A backend rule-learning AI learns the implicit rules from the demonstrations of the user (Guzdial, Li, and Riedl 2017). Our goal for this tool is that it can be used in the creation of educational, scientific, and entertaining games currently infeasible given the resource requirements of modern game development. We refer to this tool as "Mechanic Maker", as it is a variation of the existing Morai Maker (Guzdial et al. 2019).

Related Work

There have been attempts to make game development tools that are easy to use. All these approaches are unique but none of them allow for game development without programming. For example, Scratch is a "block based programming tool for creating simple games and applications" (Resnick et al. 2009). The major difference between our project and Scratch is that our tool does not require any kind of programming, making use of a backend AI to approximate the code a user would author. This is similar to visual programming (Myers 1990). However, visual programming traditionally relies on classic artificial intelligence instead of machine learning.

Other game development tools exist that draw on AI. For example, Gemini is a recent tool capable of both automatic understanding and automated generation of games (Summerville et al. 2018). Users can specify a target meaning for a game, and then Gemini uses hundreds of rules to produce a game that fits that meaning. Instead, our system employs an iterative design process, where an AI and human work together over time to produce a final game.

Other automated game design approaches incorporate AI to create a game autonomously. For example, Mike Cook's ANGELINA is an automated game designer (Cook, Colton, and Gow 2016) which designs entire games on its own. Angelina is envisioned as a 'continuous' system, not stopping in a single interaction but continually producing games. Our tool uses an AI that is also continuously on, but our AI is watching a user's actions and trying to respond appropriately to them. Shopping for game mechanics (Machado et al. 2016) is another AI powered tool which provides recommendations of existing game elements to users in order to create a game. In comparison, we learn entirely new mechanics from demonstrations. Our own prior work in automated game design produces new games based on machine learned knowledge from gameplay video (Guzdial and Riedl 2018). Similarly, our tool produces new knowledge, but now using a user's input in place of video, with a user essentially producing gameplay video for a game that does not exist.



Frame 0-3

Figure 1: Example of the Mechanic Maker UI and using the system to define simple mechanics.

Editor UI Overview

The UI of Mechanic Maker is based on the user interface (UI) of Morai Maker (Guzdial et al. 2019). However, that UI was focused on building levels collaboratively with an AI in a turn-based fashion. Comparatively, our tool is set up to allow a user to define frames of a non-existent game, while a continuously running backend AI learns to approximate mechanics that match the user's demonstrations.

We present the UI in Figure 1. On bottom left of the screen you can see the frame and input section. The frame manager identifies the current frame (frames 0 to 3 are shown in Figure 1) and arrow keys to the left and right to allow a user to go step through these frames. These frames represent a timeline of events in the user's game. Each frame is a particular moment in the game, for example the single character standing alone in Frame 3. On a frame a user can specify what that moment in a game should look like. They can then go to the next frame to demonstrate how things have changed between frames. The input buttons at the bottom left allow a user to define what input (arrow keys and space bar) are pressed in the current frame. This allows a user to specify when changes should be caused by user action. For example, clicking the space and then having a character jump from frame 0 to 1. This can also define interactions between characters as in frames 1 to 2 where two characters collide transforming them into a new character after collision in frame 3.

At the midpoint of the bottom of the screen we have a set of sprites that a user can select to place in various positions in the frame. At the far right we have a color changer for adjusting the background color of the game. The bin icon clears the screen of current frame and the gear icon loads the settings, for example specifying grid width and height. The "PLAY" button allows a user to play their game in realtime.

We note that some basic game components, like win/loss conditions, game goals, inventory, counters, UI, conditionals, etc cannot be defined in this version of Mechanic Maker, but we hope to expand it in future work.

AI Rule Learner

Our backend AI agent makes use of the same rule learning system first described by Guzdial et al. (Guzdial, Li, and Riedl 2017). Essentially, each frame of the game is re-represented as a series of percept-like components we call facts (i.e. location, sprite, velocity, and relative position information). From these facts our system learns rules, which are made up of conditions (a list of facts that must be true for the rule to fire), a pre-effect (a fact in the current frame that will be replaced), and a post-effect (the fact that replaces the pre-effect). These rules are learned through a search process, which adds, modifies, and deletes rules to a sequence of rules we call an engine. The engine begins as an empty sequence, with initial rules added based on taking every fact in a frame where a change occurred as the set of conditions required for that change to occur. These rules are not parameterized, and the sequence of rules is only limited by the input and time to search the space of possible engines. For further detail on this process please see (Guzdial, Li, and Riedl 2017).

Once we have a set of learned rules, we employ a hand-authored automated translation module to convert the rules into Unity C# code. This is relatively straightforward, as the structure of the rules already follows simple if-then rules in a sequence. Thus, during every update tick we need merely check if each of the rules fire in order.

Discussion

Our next immediate goal is a human subject study to validate the utility of Mechanic Maker as a tool for defining mechanics. We are particularly focused on users who do not otherwise have programming experience. Our plan is to compare the performance of these users to users with programming experience in a study where they attempt to reproduce a target game already created in the tool. This will allow us to compare the success rate of these different users by comparing their output games to the target. Eventually after the study we plan to make our tool more efficient by learning from all the games made by study participants, so that we can identify and add common, related rules.

Conclusions

We have presented a tool for defining game mechanics, sets of game rules visually, rather than requiring any direct programming. We hope that Mechanic maker will prove to be a useful tool in the future. In this demonstration we will showcase our initial UI and the experience of interacting with our AI rule learning system. Our plan is to continue to develop this tool towards the ability to support the development of any 2D game.

Acknowledgements

This material is based upon work supported by a start-up grant from the University of Alberta.

References

Cook, M.; Colton, S.; and Gow, J. 2016. The angelina videogame design system—part i. *IEEE Transactions on Computational Intelligence and AI in Games* 9(2):192–203.

Guzdial, M., and Riedl, M. 2018. Automated game design via conceptual expansion. In *Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Guzdial, M.; Liao, N.; Chen, J.; Chen, S.-Y.; Shah, S.; Shah, V.; Reno, J.; Smith, G.; and Riedl, M. O. 2019. Friend, collaborator, student, manager: How design of an ai-driven game level editor affects creators. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 1–13.

Guzdial, M.; Li, B.; and Riedl, M. O. 2017. Game engine learning from video. In *IJCAI*, 3707–3713.

Machado, T.; Bravi, I.; Wang, Z.; Nealen, A.; and Togelius, J. 2016. Shopping for game mechanics.

Myers, B. A. 1990. Taxonomies of visual programming and program visualization. *Journal of Visual Languages & Computing* 1(1):97–123.

Resnick, M.; Maloney, J.; Monroy-Hernández, A.; Rusk, N.; Eastmond, E.; Brennan, K.; Millner, A.; Rosenbaum, E.; Silver, J.; Silverman, B.; et al. 2009. Scratch: programming for all. *Communications of the ACM* 52(11):60–67.

Summerville, A.; Martens, C.; Samuel, B.; Osborn, J.; Wardrip-Fruin, N.; and Mateas, M. 2018. Gemini: Bidirectional generation and analysis of games via asp. In *Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference*.