MiniStS: A Testbed for Dynamic Rule Exploration

Bahar Bateni[1], Jim Whitehead[1]

[1]University of California, Santa Cruz (UCSC), Santa Cruz, CA 95064

Abstract

Specialized testbeds are valuable tools in game AI and procedural content generation (PCG) research, providing controlled environments to test and evaluate new algorithms and explore game mechanics. This paper introduces MiniStS, a research testbed modeled after the rogue-like card game Slay the Spire (StS), tailored for studying game-playing AI and procedural generation. MiniStS features a dynamic rule system where in-game cards can modify game rules, providing a rich context for research. We argue that such an environment offers distinct research opportunities by revealing insights into game design, enabling the exploration of rule combinations and synergies, and advancing AI capabilities. We define possible applications for MiniStS in terms of generalized game-playing that requires understanding and adaptability to the dynamic changes to the rules, and card generation with a focus on a deepened exploration of rule synergies.

Keywords

Collectible Card Games, General Game-playing, Automated Game Design, Procedural Content Generation

1. Introduction

Game AI and procedural content generation (PCG) research frequently benefits from specialized testbed environments that are tailored to specific research needs. These environments provide controlled settings where new algorithms can be developed and evaluated. Furthermore, they can enable an examination of game mechanics and design principles.

One prominent example of this is Infinite Mario Bros [1], a research testbed designed to emulate the classic platformer *Super Mario Bros.* Infinite Mario Bros has enabled researchers to explore areas such as level generation [2, 3, 4, 5], game-playing AI [6, 7], accessibility [8], visualization of playtesting data [9], and, venturing outside of games, software repair [10]. GVGAI [11] is another testbed environment which implements a variety of 2D arcade games, such as *Space Invaders*, *Pong*, *Solarfox*, and more. Other research environments include implementations of *Angry Birds* [12], *Overcooked* [13], and *Baba is You* [14].

In this paper, we introduce MiniStS,[1] a simplified Python implementation of the rogue-like card game *Slay the Spire* (StS) [15]. MiniStS serves as a dedicated research environment for investigating various aspects of game AI and procedural generation of cards. Similar to other collectible card games, *Slay the Spire* includes cards that can affect the design of the game by enabling or modifying certain rules of the game. Because of this, it demonstrates characteristics that allow interesting research avenues.

One of these characteristics is that a game-playing agent in this game is required to have reasoning capabilities related to game design, since it needs to manipulate these playable rules to its own advantage. Further, agents should be generalized enough to play the game while rules are dynamically changing and adapt its strategies accordingly. As for a PCG agent, it is necessary to not fixate on the context of a single ruleset, but to consider all the possibilities arising from combining different rules in different ways. We discuss these properties and more in Section 1.2.

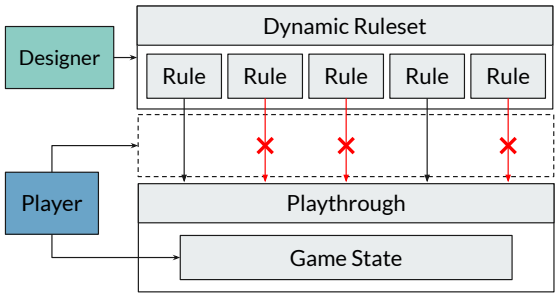

Figure 1: The structure of a game with a dynamic rule system. In such a game, the designer defines a set of possible rules or design elements in the game that can be combined together. The game then allows the player to manipulate the design of the game through playing the game and activating certain rules.

1.1. Dynamic Rule System Definition

We define games with dynamic rule systems as games in which the rules can be changed by playing the game. One of the most interesting examples of this is the game *Baba is you* [16]. In this game, the player can move blocks around a grid-based level in a *Sokoban* style game. In contrast with *Sokoban*, here the blocks can also represents game design concepts and elements in the level. For example, if the player moves the three blocks representing "Flag", "Is" and "Win" together, it results in activating a rule in the game were the player touching the flag will result in winning the level.

Baba is You is not the only game that utilizes a dynamic rule system. Collectible card games often include cards which have game design implications. These cards can enable or modify the rules of the game to some extent either temporarily or for the duration of the battle.[2,3] In these games, the players are invited to think about the consequences of activating or modifying different rules in the game and use these changes strategically to their advantage.

Another genre of games that have many examples of using a dynamic rule system are roguelikes. The definition

11th Experimental Artificial Intelligence in Games Workshop, November 19, 2024, Lexington, Kentucky, USA.

bbateni@ucsc.edu (B. Bateni); ejw@ucsc.edu (J. Whitehead)
https://bahar.bateni.org (B. Bateni); https://users.soe.ucsc.edu/~ejw/ (J. Whitehead)
0000-0002-0701-0311 (B. Bateni); 0000-0002-6887-7330 (J. Whitehead)

[1]The code is available at https://github.com/iambb5445/MiniSTS

[2]An example of such a card in *Hearthstone* is *Cold Feet* which causes enemy minions to cost 5 more next turn. This card modifies the initial rule of "Card A costs X" for one turn.

[3]An example of such a card in *Magic: the Gathering* is *Elder Mastery* which enchants a creature so that whenever it deals damage to a player, that player is forced to discard two cards. This cards adds a new rule to the game for a certain creature.

of a roguelike game has been a point of debate in the community.[4] However, one of the main goals of this genre is a sense of exploration and a gameplay that varies every run. Toward this goal, there are numerous examples of roguelikes that include rule-altering items in the game. The game uses these alterations to not only provide a strategic aspect to the game, but also enable various possibilities in the way the game is played. Examples of this type of gameplay can be seen in roguelikes such as *the Binding of Isaac* [18], *Hades* [19], *Noita* [20], *Risk of Rain 2* [21] and more.

One important note is that the distinction between games with static rule systems and dynamic rule systems is not always pronounced. For example, a first person shooter (FPS) can allow the player to choose between weapons such as an sniper or an assault riffle, which affects the design of the game and subsequently the way the game is played. However, the extent of these changes and the variety of the possible designs that can be created by these choices are more limited. Dynamic rule systems specially involve rules that can be combined together and create a wide variety of unique selections. Additionally, games with dynamic rule systems rely on the changes to the rules as one of the main components of the game.

1.2. Dynamic Rule System Research

In the previous section, we describe a dynamic rule system and the genres that frequently use such systems. In this section, we argue that these games can provide unique opportunities for research on games.

First, games with dynamic rule systems can help us to discover novel viewpoints in understanding game design. For example, in *Baba is Y'all* [14], the authors describe how "the game provides an interesting study for procedural level generation where game levels and game rules are intertwined." Not only does this introduce a new challenge to level design, where the game design and its possible outcomes should be considered when designing the levels, but it also suggests the possibility that this connection could have been explored deeper in other games, even ones with static rule systems.

Second, dynamic rule systems can enable the study of rule combinations in a way that cannot be explored in a static rule system. As shown in Figure 2, in a dynamic rule system the set of available rules creates a space of possible designs in which every valid point is the result of combining rules by activating them in some order (in a card game, via playing cards). The player choices can then result in moving in this space from one point to the other, resulting in a different experience every time.

When taking the role of the designer and adding a new rule to the set of available rules, the effect of this new rule should be examined on all the possible designs from every possible combination. This effect can move the design points in the space and result in a distribution of design points with a lower or higher design quality according to the relevant research metrics. Figure 3 visualizes this characteristic.

This study of rule combinations and synergies is not limited to when an AI assumes the role of the designer. Similarly, when developing a game-playing AI, it is necessary

[4]One popular interpretation of roguelike is the Berlin Interpretation, [17] which defines a set of characteristics expected from roguelikes. However, many of these characteristics, such as turn-based, grid-based, and ASCII art style, do not apply to a significant number of modern roguelikes.

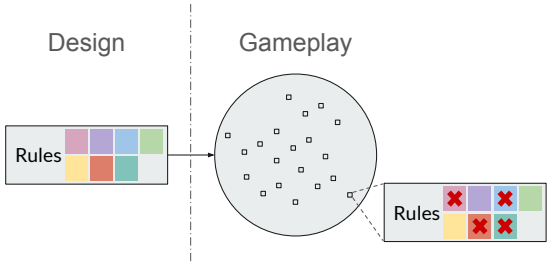

Figure 2: The design space resulting from a single set of rules in a dynamic rule system. The designer initially defines a set of composable rules in the game during the game design process. The player then dynamically changes the set of active rules during gameplay by making in-game decisions. As a result, the player to some extent collaborates on game design decisions.

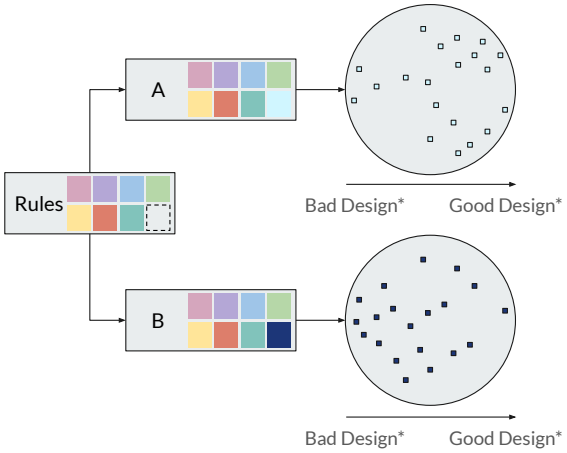

Figure 3: Exploring the effect of adding rule A vs rule B in a dynamic design system. The effect of adding each rule should be viewed in terms of how it affects the space of valid combinations in the design space as opposed to in a single fixed ruleset. The good design and bad design visualizations are assumed according to system's design quality metric.

for the agent to be able to understand this space to some extent in order to steer toward designs that are better suited for the agent's goals. The challenge for game-playing AI is to reason about what effects its decisions have on the game since these decisions relate to game design.

Finally, it's important to note that research on dynamic rule systems can be beneficial for improving the development of such games. In other words, such research has beneficial impacts not only on game design and game-playing AI, but also on the development of new games with dynamic design systems. In roguelikes, random generation and PCG systems are often used to create a different experience every playthrough and foster a sense of exploration in the player. A PCG system designed specifically for dynamic rule system can enhance this design concept by ensuring that the procedural content aligns more closely with the evolving gameplay, or expand the set of available rules by finding rules that fit its malleable rule system.

In a similar fashion, the interactions between dynamically added rules is important in many games, especially card games like MiniStS. When the cards are designed in a way to enable various synergies and interactions between the cards, they enable lenticular design [22], a term introduced by Mark Rosewater, the head designer of *Magic the Gathering*.

Lenticular design is when understanding a card is more difficult for an experienced player compared to a beginner. If the cards include many synergy effects when combined in certain ways, an experienced player would likely have to consider all of these effects when playing. On the other hand, a novice player would not understand these effects and hence would not get lost in the complexity of the game. This is a desirable effect for game designers, since it means that the barrier to entry is low allowing novice players to feel comfortable playing the game with minimal instructions, while experienced players continue to find new layers of complexity to the game.

1.3. *Slay the Spire*

Slay the Spire is a single-player roguelike card game that has a strong emphasis on strategic play and card synergies. We chose this game to implement as the testbed environment since it has desirable characteristics from both roguelikes and collectible card games.

First, as with many roguelike games, StS is a single-player game. The game is played against a set of pre-defined enemies that perform different moves with different probabilities independent of the player actions. Therefore, the challenges of understanding the opponent skill level in evaluating game-playing capabilities of an agent can be eliminated in favor of focusing on an understanding of rules and their combinations, both during game-play and during design of these rules. Similarly, the strength of different cards or decks can be evaluated without considering how well they perform against different opposing decks.

Furthermore, the core loop of the game allows for an incremental approach to deck-building. At the end of every battle in StS, the player is asked to choose at most one of three possible cards to add to their deck. The increasing level of difficulty for each battle and the enemies are carefully designed in a way that appropriately matches bigger and more powerful decks as the game progresses. Because of this, a game-playing agent defined in this environment doesn't need to necessarily be exposed to the challenges that arise from the vast space of possibilities for possible decks. Instead, the agent can be tested in terms of combining rules from a single set of cards in a pre-defined deck, and the game offers various examples of well-designed levels with different degrees of difficulty suited for this evaluation.

Additionally, similar to most collectible card games, *Slay the Spire* is turn-based. As a result, game-playing in this environment can be done without any considerations of real-time play and mimicking human players in terms of reaction time. The possibility of eliminating these challenges in StS is desirable since it makes the MiniStS environment better suited for our goals.

2. Related Work

Research in games often utilize game environments that implement existing games to explore various aspects of game design, level generation, and artificial intelligence. As mentioned earlier, one notable example is Infinite Mario Bros (IMB), a public domain clone of the classic *Super Mario Bros.* IMB incorporates a simple procedural level generation process that combines pre-defined segments to create a level. One of the ways that researchers have used IMB is by experimenting with different level generation methods such as applying rhythm-based approaches [2], creating level sections focused on certain mechanics [3], or composing predefined level sections [4, 5].

In addition to its applications in level generation, IMB has also served as a gameplaying environment acting as a benchmark for artificial intelligence research [7]. This resulted in the introduction of the Mario AI Framework [6], which has been utilized in AI competitions to evaluate and compare various AI strategies. Beyond these applications, IMB has also been used in visualizing and analyzing play-testing data [9], exploring accessibility methods [8], and contributing to fields outside of game studies, such as runtime software repair [10].

Another research game environment is the physics-based game *Angry Birds*. Since the game *Angry Birds* does not have an open source implementation, Ferreira and Toledo implemented a clone of the game for the purpose of exploring level generation in this game [12]. This environment, named Science Birds, has been used to explore a variety of level generation approaches such as procedurally constructing and combining structures [23], utilizing Generative Adversarial Networks (GANs) for level generation [24], and using genetic algorithms [24]. Furthermore, Science Birds has been selected as the platform for the GPT4PCG [25] competition in which the use of Large Language Models in generating levels via effective prompt-engineering is explored. The AIBirds [26] competition also uses *Angry Birds* as a testbed for both level generation and game-playing, and uses Science Birds for its level generation track.

The IMB and Science Birds testbeds are not the only examples of re-implementing games for research. Hu et al. list other examples of environments developed for research purposes based on video games [27], some of which have been significantly simplified to better fit the intended purpose. These examples include a simplified version of *Overcooked* developed to experiment with cooperative abilities of AI agents [13], a minimal implementation of Real-Time Strategy (RTS) game mechanics used for microRTS game AI competition [28], and an implementation of *the Legend of Zelda: a Link to the Past* developed specifically as a PCG research environment [29]. It's also worth noting that some games such as *Starcraft II* provide an API for interacting with the game which can be used to develop game-playing agents that can directly interact with the game via these APIs [30].

General Video Game AI (GVGAI) [11] is an example of a general purpose environment that provides a wide variety of 2D arcade games as a testbed for evaluating AI agents in terms of general video game playing capabilities. The agents in this environment are expected to not be game-specific, which requires them to be able to play games that they have not been trained on. Similar to this goal, MiniStS also expects the agents to be able to adapt to changes in the rules of the game. However the StS environment requires the game-playing AI agent itself to actively make changes to the rules through playing the game. This is an important difference that requires the agent to reason about how a specific rule would benefit or harm its chance of winning, and how best the rules can be composed to fit the agent's goals. Furthermore, the card game nature of StS allows for the possibility of an agent understanding these rules via language-based reasoning. Our previous work [31] has shown that such an agent in this environment can show long-term reasoning capabilities required in some scenarios.

MiniStS is not the only card game environment avail-

able for research purposes. XMage[5] and Hearthbreaker[6] are community-made clones of *Magic: the Gathering* and *Hearthstone* respectively. Ling et al. use the dataset of cards from both of these environments to perform research on code generation [32]. Metastone[7] is a community-made implementation of Hearthstone which Santos et al. use to explore use of the MCTS algorithm in playing Hearthstone [33]. One important difference between MiniStS and the community-made environments is our focus on not limiting MiniStS to the set of existing cards. Since our goal is enabling research that explores dynamic rule systems and an understanding and adaption to the rules during gameplay, we specifically design MiniStS to facilitate these goals.

Chen and Guy introduce Chaos Cards [34] as a card generation system. They use the Chaos Card environment, which has similar rules to *Hearthstone*, to both generate playable cards and playtest them by using a 1-step look-ahead agent. The cards are designed via a specific grammar that runs through GIGL [35], which was previously made by the same authors to enable creating executable artifacts from grammatical descriptions. While this environment is well-suited for the purposes of the authors, our focus lies on cards that allow changes to the rules and can possibly have effects beyond the Hearthstone-specific grammar. Additionally, MiniStS also removes some of the challenges of playing against an opponent and deck-building to enable a more direct focus on studying the underlying dynamic rule system, which we discussed in Section 1.3.

Finally, while not a card game, *Baba is You* has similarities with MiniStS in terms of qualities of the environment. As discussed in Section 1, *Baba is You* is an interesting research subject since, similar to games with as dynamic rule systems such as StS, it provides an environment that the player interacts with the rules and changes them during gameplay. Not only does this create interesting challenges in level design and intertwines level design and game rules, it also raises interesting questions in terms of game-playing that relates to the AI agent's understanding of how to best play with the rules. Charity et al. use an implementation of the game based on the available code for the initial game jam version of the game to research their proposed collaborative mixed-initiative design system [14].

3. Applications

In this section, we highlight two types of application for which MiniStS can be used. First, we can use MiniStS to explore dynamic rule-playing agents, i.e. game-playing agents that can act in a dynamic rule system, deciding how to play the game in ways that possibly result in a change of rules. The agents should be able to continuously adapt to these changes. Furthermore, the agents should have an understanding of game design and the effect of rules on the game to be able to choose how to manipulate the active rules toward their goal.

Second, we discuss the use of MiniStS as a card generation environment. Since MiniStS is designed in a way to allow the definition of new cards by combining existing actions and effects or defining new ones, this environment can be used for designing and playtesting new cards.

[5]https://github.com/magefree/mage/
[6]https://github.com/danielyule/hearthbreaker/
[7]https://github.com/demilich1/metastone

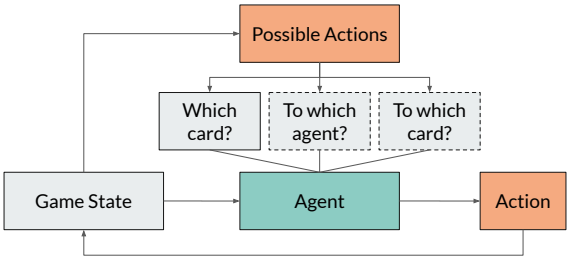

Figure 4: The API required for defining a rule-playing agent in MiniStS. The agent should be able to respond to three type of questions given the state of the game and the list of possible responses.

Though these two applications can be explored independently, they relate to each other in an interesting way. The dynamic rule-playing agent can be used as a means of evaluation to playtest the game when new cards are introduced to the game. On the other hand, defining new cards can test the agents in terms of generalization since the agent can be provided with new rules to play with without being trained on those specific set of rules. A virtuous cycle.

Below we describe the required API to tackle each of these challenges, as well as our existing examples of agents that respond to each target application.

3.1. Dynamic Rule-Playing AI

In MiniStS, we define the term game-playing agent (GPA) as an agent that take the role of the player in the game by making any decision that is expected from the player. Any GPA in MiniStS is required to implement three functionalities:

1. Choose a card to play: Given the current state, the agent should be able to decide which one of the available cards to play, or decide to end its turn.
2. Choose the target (agent): Some of the cards can target one or more agents in the game, for example the player should choose the target for "Strike: Deal 6 damage." Given the state of the game and a list of possible options, the GPA should be able to decide which option(s) are the target of the card.
3. Choose the target (card): Similarly, some cards can target other cards, such as "Concentrate: Discard 3 cards. Gain 2 energy." Given the state of the game and a list of possible options, the GPA should be able to decide which option(s) are the target of the card.

The only requirement for playing the game is for a GPA to be able to respond to these three questions. MiniStS also allows the agents to simulate the future steps of the game, making it possible to define agents that use this prediction in their decision making, such as an MCTS agent. Note that this simulation is not completely accurate, since any non-deterministic actions (e.g. shuffling the deck, random choice, etc.) are intentionally forced to use a different seed. This is because none of the agents should have perfect knowledge about the non-deterministic events of the game.

In our previous research [31], we developed three different agents within MiniStS and compared them in different scenarios. First, we showed that a random agent can be defined by simply choosing one of the options randomly for each of these questions, as shown in Figure 5. Second, we utilized the simulation abilities of MiniStS to showcase

```python
class RandomAgent(GGPAgent):
    def choose_card(self, game_state) -> EndTurn|PlayCard:
        options = self.get_options(game_state)
        return random.choice(options)

    def choose_agent_target(self, game_state, agents) -> Agent:
        return random.choice(agents)

    def choose_card_target(self, game_state, cards) -> Card:
        return random.choice(cards)
```

Figure 5: Pseudocode showing implementation of an example agent (random agent). The agent implements functions to decide the next action based on the state of the game and to choose a target between agents/cards when needed. The pseudocode is slightly simplified from the actual implementation.

a look-ahead agent. The look-ahead agent simulates the game up to a pre-defined depth and chooses an action for the current move that would maximize the estimated score in the future. Finally, we demonstrated how MiniStS can be used with a language-based agent by converting the game state into a purely text-based format and sending it to a large language model (LLM) along with the basic rules of the game. The text-based description for each card is automatically generated based on its effects but can also be customized to show a predefined text. All three agents are generalized, meaning that they are not limited to playing only a specific set of cards. These three agents are included in MiniStS's source code and can be used to run simulations in different settings. Our primary focus in this paper is to present MiniStS as a testbed, rather than the agents. Our previous work [31] provides an example of how MiniStS can be used to simulate and compare different agents in a dynamic rule system, and includes more information on each of these agents.

The testbed that MiniStS provides is uniquely useful in testing rule-playing agents. As discussed in Section 1, one of our main goals with MiniStS is to define an environment in which playing the game can change the way the game is played. As such, we call the game-playing agents in this environment rule-playing agents, since the agent is playing with the rules. In other words, the performance of such an agent is tied to its understanding of how these rules change the game, and whether or not it would benefit from these changes.

MiniStS also includes an agent evaluation module that allows comparing different agents given a scenario. The scenario defines which cards the agents have in their deck, and what enemies are they encountering. The evaluation component then simulates multiple instances of the game in parallel and logs the game events and the response of agents to these events. Finally, these logs are imported by another module to calculate a set of metrics such as win-rate, the distribution of player's health at the end of the game, whether a certain card or combinations of cards were used by the agent, how many turns the agent took to win/lose the game, and so on.

The execution time of the simulation is highly dependent on the number of turns it takes for the agent to end the game and the agent's performance. To provide an estimate of MiniStS's performance and factor out the time it takes for the agent to make decisions, we can consider an agent that always chooses the first available action. In this case, the first battle of the game, using the starting set of cards, averages around 5 turns and 23 agent actions (calculated over 1,000 simulations). Running these 1,000 simulations on a personal laptop with an RTX 3050 GPU and on 50 threads takes less than 10 milliseconds per simulation on average.

3.2. Card Generation

The second goal of MiniStS is to enable researchers to define card-generation systems. As we discussed previously, when the cards can alter the rules of the game, card generation requires a careful consideration of rule synergies and the space of the game design. To achieve this goal, we design the system in a way to allow versatile combination of different actions and effects. These actions and effects can be from the existing actions, but a major part of our focus is in not limiting our system to a pre-defined set of actions. The actions can also be combined with an AND module. If the action requires a target, MiniStS forces the card definition to include the TO module for defining how this target is selected. We discuss these modules in more detail and with examples in Section 4.

Additionally, some cards can change the rules to have specific effects when certain events occur. To make this possible, we define status effects that visualize these changes in the rules, similar to how the game visualizes these changes for the player. We also use an open-ended event system that allows for defining broadcast events and subscribing to them to perform actions that result from the rule change. We explain MiniStS components, such as actions, their targets, status effects, and event system in more detail in Section 4.

With this setup, creating a new card in the game can be done by following these steps:

1. Use the set of existing actions and define any new actions that are required. Each action can access the state of the game and manipulate it. However, the actions should be defined as atomic operations, meaning that they are small enough that cannot be broken down further into lower level actions.
2. If there are any new status effect, define these effects and determine their properties, such as if multiple instances of the same effect would stack, if the effect is for a certain number of turns, and so on.
3. If the card requires any new events or callback on the existing events, define the new modules.
4. Combine the actions, and determine any required card parameters (such as cost of the card) or action parameters (such as amount of damage for the action "Deal x damage").

Example of card definitions and combining the actions is available in Section 4.

To show a simple example of card generation, we implement a simple grammar-based card generation module. The card generation module randomly chooses actions and possibly combines them by using the AND module. The parameters for each action are chosen from a set of reasonable values that are determined based on examples of existing cards. The card generation process lacks any evaluation of the quality of cards and is also limited to the set of available actions and their combinations, and does not introduce new rules into the game. We use the available card generation process not as a strong method for generating cards, but as both an example of how cards can be generated from these actions and to enable a simple way of providing new cards for the purpose of testing rule-playing agents when playing with never-seen-before cards.

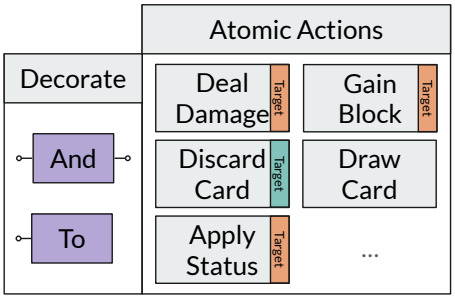

Figure 6: Atomic actions. Some actions have a specific type of target (agent or card) and can be decorated with TO. Any action or combination of actions can be decorated with AND.

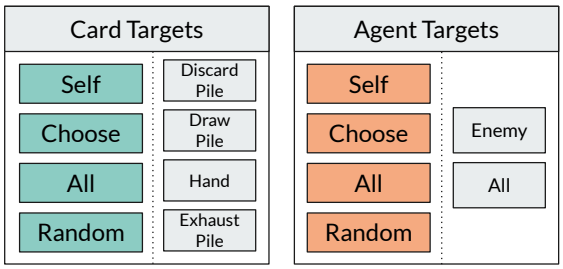

Figure 7: Possible targets for atomic actions. The targets can be either a card or an agent (i.e. the player or enemies). Each of these targets is defined on a set of possible options, and a way of choosing the option.

4. System Structure

In this section, we describe different components of the MiniStS design with the goal of creating a research environment with a dynamic rule system.

One of our main goals with MiniStS is to enable and streamline the process of defining a new card. To achieve this, we analyze a set of existing cards, both from Slay the Spire and from a set of third party cards designed as mods by the StS community.

4.1. Actions

The first component that we identify in the cards are actions. Actions define what effect the card has on the state of the game. For example, a "Gain Mana" action means that when the card is played, it changes the amount of available mana for the player. Some of these actions are visualized in Figure 6.

The actions are defined as atomic, meaning that when we analyze a card, we break down its effect into indivisible actions that cannot be further decomposed. We then allow these atomic actions to be combined together in any order by defining an AND decorator that creates an action from the combination of any two actions.

Finally, many actions in the game require one or more targets. For example, the "Deal damage" action affects one or more enemies which are selected in a specific way. We make this possible by defining actions that need a target. For each of these actions, the designer should specify how the required targets are selected, whether based on player input, random selection, or chosen as "all the possible targets." These actions are considered incomplete unless they are decorated with TO and a target definition. We discuss the target definitions in the next subsection.

Note that the AND decorator can still be applied before or after defining a target. Figure 9 shows pseudocode for two example cards, and Figure 10 shows how the second card can be redefine to apply its underlying actions to two, possibly different, targets. Figure 8 visualizes the two example cards.

4.2. Targets

As mentioned previously, the effect of a card can be expressed by a set of atomic actions, some of which require a target. By analyzing the existing cards, we see that StS cards can target agents (i.e. the player or the enemies) or cards. Each of these targets can be expressed as one of the following types:

- Self: Either the player or the card that includes this action, based on the type of target (agent or card).
- Choose [from a set of options]: The target is identified based on player input.
- All [from a set of options]: The action with this type of target is applied to all the the possible target options.
- Random [from a set of options]: The target is chosen as one or more options from the set of available options.

For each of these cases, the set of available options can be defined in a few different ways. By examining the existing cards, we identify the possible set of options for card targets to be either from player's hand, discard pile, draw pile or exhaust pile. As for the agent, if the target is not the player (self), it's from the enemies. We also noticed in a set of third party cards from community mod packs that the target can also refer to all the agents (i.e. both the player and the enemies).

4.3. Status Effects and Events

Slay the Spire defines a set of status effects, or buffs and debuffs, which can be used to indicate when an agent is weak (deals less damage for a number of turns), vulnerable (takes more damage for a number of turns), etc. While we can implement only these effects by using specialized code that only covers these cases, we want the definition of a status effect to be more open-ended and include a flexible infrastructure, since status effects are the foundation of rule-based cards.

So far, we have defined atomic actions and targets which enable many of the available cards in the game to be expressed in MiniStS. However, many of the most interesting cards in the game, especially the cards that change the rules of the game, cannot be defined with the actions alone. For example, the card "After Image" is defined as "Whenever you play a card, gain 1 block." Instead of having an instant effect on the state of the game, these cards can have a lasting effect that can be triggered at certain points, such as whenever a card is played, at the start of every turn, until the end of this turn, etc.

To understand how best we can cover these types of effects in MiniStS, we first look at how StS communicates these cards and their effects with the player. StS uses status effects to visualize these changes to the rules of the game. For example, the card "After Image", which we discussed above, applies a specialized status called "After Image" on

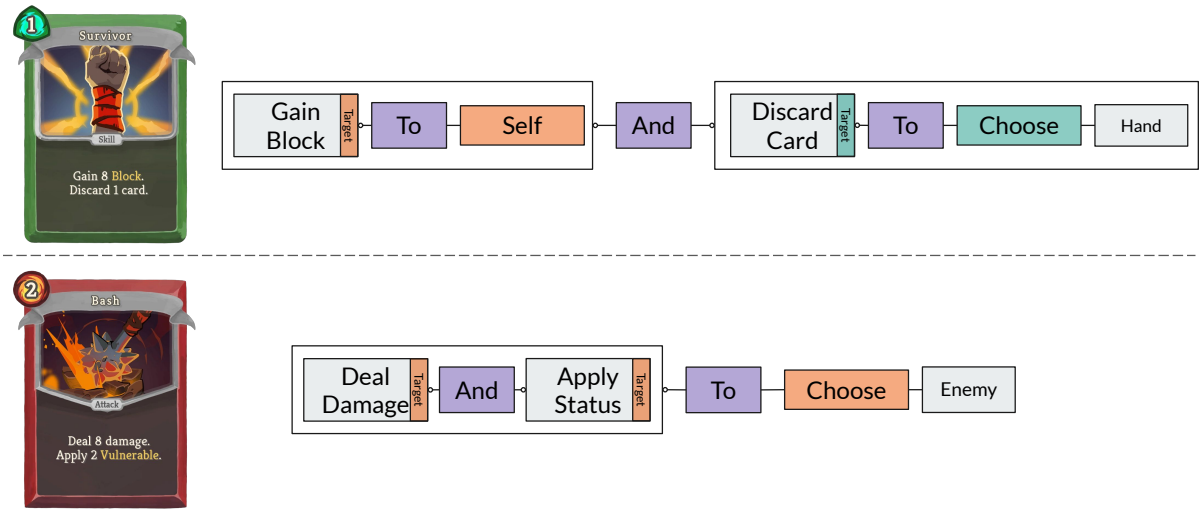

Figure 8: visualization of implementing "Survivor: Gain 8 Block. Discard 1 card." and "Bash: Deal 8 Damage. Apply 2 Vulnerable." Both cards can be defined by combining atomic actions and targets with AND and TO. However, Bash uses the same target for both actions, which requires using AND before defining the targets with TO, whereas Survivor have different targets and uses AND after seperately defining the targets.

```
Card("Survivor",
    AddBlock(8).To(SelfAgentTarget()).And(
    DiscardCard().To(ChooseCardTarget(CardPile.HAND)))
)

Card("Bash",
    DealAttackDamage(8).And(
    ApplyStatus(2, VULNERABLE)).To(
        ChooseAgentTarget(AgentSet.ENEMY)
    )
)
```

Figure 9: Pseudocode showing implementation of two example cards, "Survivor: Gain 8 Block. Discard 1 card." and "Bash: Deal 8 Damage. Apply 2 Vulnerable." The pseudocode is simplified from the actual implementation.

```
Card("Bash*",
    DealAttackDamage(8).To(
        ChooseAgentTarget(AgentSet.ENEMY)
    ).And(
    ApplyStatus(2, VULNERABLE)).To(
        ChooseAgentTarget(AgentSet.ENEMY)
    )
)
```

Figure 10: Pseudocode showing implementation of an imaginary card, "Bash*", which deals 8 damage to one target, and then applies 2 Vulnerable to another, possibly different, target. The pseudocode is simplified from the actual implementation.

the player. Each effect also has a set of properties. For example, “After Image” status effect does not decrease during the battle, and can stack (i.e. if “After Image” is played twice, the value of status effect is increased to two).

Finally, the status effect performs some behavior when the appropriate event triggers. To implement this, we use an event system. The event system provides a formal definition of how the events and callbacks related to status effects should be added to the game. Each event has broadcast and subscribe functions. Broadcast is used by the entity that relates to the condition of the event, while subscribe is used

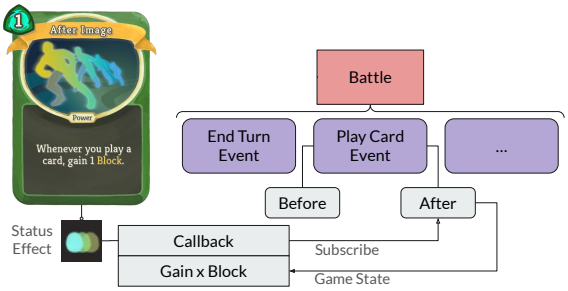

Figure 11: Definition of card “After Image” in MiniStS. The applies a special status effect, called “After Image”, to the player. The unique callback is subscribed to the event of “Play Card” to trigger after every time it happens. The callback then, given the whole state of the game, applies some amount of block to the player depending on the strength of the status effect which directly maps to how many times the card was played.

to register the callback to that event. The callbacks can be subscribed to trigger before or after an event. Figure 11 demonstrates how status effects and the event system can be used to create “After Image.”

The events can also be defined in a way to manipulate a value. For example, the status effect “vulnerable” makes it so that the affected creature takes 50% more attack damage for some number of turns. The vulnerable status effect is implemented by subscribing to the “attack damage” event. Basically, whenever the “deal attack damage” action is performed in the game, the event triggers and expects the callbacks subscribed to it to manipulate the amount of damage. Each event system can have any type of input that it desired, or not have an input at all. Each callback is then required to specifically get that type of input and returns the manipulated value.

Finally, multiple callbacks can be defined on each event. For example, “weak” and “vulnerable” effects are both defined to trigger when attack damage is happening in the game and manipulate its value. However, the order in which

these effects apply completely depend on how the card designer have envisioned the effects. Because of this, it's important for any new callback to be subscribed at the correct order compared to other callbacks.

4.4. Gameplay Loop

The gameplay loop simulates the events of a battle. The event and values of this component are a simulation of the events in Slay the Spire. Each turn consists of the following events:

1. Gain Mana: The player gains full mana at the start of the turn.
2. Draw a Hand: The player draws cards equal to the number of pre-defined draw count. If at any times the draw pile is empty, the discard pile is shuffled back into the draw pile.
3. Play (Player Side): The player side is played, meaning that the player agent would take its turn, and then the enemy side would lose any remaining block. Note that some damage types would apply after this event, bypassing enemies' blocks.
4. Play (Enemy Side): The enemy side is played, meaning that the enemies would take their turn one by one, and then the player side would lose any remaining block.
5. Discard Hand: The cards in player's hand are then discarded and moved to the discard pile.

The game will continue repeating the events described above every turn until the battle has ended, i.e. the player or all the enemies are dead. Note that these events can be affected by the cards, e.g. a card can change the number of draw count, make the cards not get discarded at the end of the turn, etc.

The gameplay loop is also connected to the game state such as the available deck of cards, properties such as draw count or maximum mana, and state of the battle including a the list of cards in player's hand, discard pile, draw pile and exhaust pile, and the player and enemies and their respective states such as HP, mana, block, and so on.

5. Limitations

One limitation of the system is that it respects the basic rules of the game *Slay the Spire*, so any changes that are related to the foundation of the game would require changing the system. For example, the order of events (e.g. the player plays first, then the enemies take their actions), the existence of only a single type of mana, or the fact that this is a single player card game cannot be modified without applying significant changes to the code.

Another limitation of the system is regarding the types of input the system expects from the system. As discussed in Section 4.2, the game expects each action to be applied to at most one agent or card. Because of this, any action that requires a new type of input does not exist in the game. For example, one can imagine a card defined as "Remove one of the status effects of your choosing" which would require a new form of input from the player. Furthermore, actions that are defined on multiple target types do not exist in StS or MiniStS, such as "Discard a card and deal damage equal to 5 times its cost." It's worth noting that this still can be implemented as combination of two actions by keeping track of the cost in the game state.

StS includes multiple characters, some of which have specific mechanics. The current implementation of the game, which is a simplified version, does not implement these mechanics. Furthermore, the current version of MiniStS implements some but not all the existing actions, status effects, and enemies in the game, and only simulates one battle to compare agents instead of multiple battles. However, such components can be added to MiniStS without any major changes to the code.

6. Conclusion

Games with dynamic rule systems are games in which a set of composable rules exist which can be activated through playing the game. While the designer is the one who creates these rules, the player also participates in game design decisions by playing the game. This unique property, a key characteristic of collectible card game and roguelike genres, offers unique research opportunities. AI agents can be cast into the role of the designer or the player in the system. As a designer, these agents are required to reason about the synergy effect of rules and the distribution of the possible designs that arise from different rules activated in each playthrough. As a player, these agents are required to reason about game design in order to combine the rules in a way that gets them closer to their goal.

In this paper, we introduced MiniStS, a simplified implementation of *Slay the Spire*. MiniStS enables researchers to explore the StS environment and its dynamic rule system in two ways. First, with the role of a player in the system and as a rule-playing agent, and second, with the role of a designer and as a card generation task. We describe the steps with which both of these tasks can be approached in MiniStS. Finally, we described different components of MiniStS and discussed how each of these components facilitate these two applications by making it possible to define new cards and rule-playing agent without any major changes to the code and by using a streamlined process.

References

[1] M. Persson, Infinite mario bros, Online Game (2008).

[2] G. Smith, M. Treanor, J. Whitehead, M. Mateas, Rhythm-based level generation for 2d platformers, in: Proceedings of the 4th International Conference on Foundations of Digital Games, FDG '09, Association for Computing Machinery, New York, NY, USA, 2009, p. 175–182. URL: https://doi.org/10.1145/1536513.1536548. doi:10.1145/1536513.1536548.

[3] A. Khalifa, M. C. Green, G. Barros, J. Togelius, Intentional computational level design, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '19, Association for Computing Machinery, New York, NY, USA, 2019, p. 796–803. URL: https://doi.org/10.1145/3321707.3321849. doi:10.1145/3321707.3321849.

[4] M. Cerny Green, L. Mugrai, A. Khalifa, J. Togelius, Mario level generation from mechanics using scene stitching, in: 2020 IEEE Conference on Games (CoG), 2020, pp. 49–56. doi:10.1109/CoG47356.2020.9231692.

[5] P. Mawhorter, M. Mateas, Procedural level generation using occupancy-regulated extension, in: Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games, 2010, pp. 351–358. doi:10.1109/ITW.2010.5593333.

[6] S. Karakovskiy, J. Togelius, The mario ai benchmark and competitions, IEEE Transactions on Computational Intelligence and AI in Games 4 (2012) 55–67. doi:10.1109/TCIAIG.2012.2188528.

[7] J. Togelius, S. Karakovskiy, J. Koutnik, J. Schmidhuber, Super mario evolution, in: 2009 IEEE Symposium on Computational Intelligence and Games, 2009, pp. 156–161. doi:10.1109/CIG.2009.5286481.

[8] J. Muñoz, G. N. Yannakakis, F. Mulvey, D. W. Hansen, G. Gutierrez, A. Sanchis, Towards gaze-controlled platform games, in: 2011 IEEE Conference on Computational Intelligence and Games (CIG'11), 2011, pp. 47–54. doi:10.1109/CIG.2011.6031988.

[9] G. Wallner, N. Halabi, P. Mirza-Babaei, Aggregated visualization of playtesting data, in: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI '19, Association for Computing Machinery, New York, NY, USA, 2019, p. 1–12. URL: https://doi.org/10.1145/3290605.3300593. doi:10.1145/3290605.3300593.

[10] C. Lewis, J. Whitehead, Runtime repair of software faults using event-driven monitoring, in: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2, ICSE '10, Association for Computing Machinery, New York, NY, USA, 2010, p. 275–280. URL: https://doi.org/10.1145/1810295.1810352. doi:10.1145/1810295.1810352.

[11] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, S. Lucas, General video game ai: Competition, challenges and opportunities, Proceedings of the AAAI Conference on Artificial Intelligence 30 (2016). URL: https://ojs.aaai.org/index.php/AAAI/article/view/9869. doi:10.1609/aaai.v30i1.9869.

[12] L. Ferreira, C. Toledo, A search-based approach for generating angry birds levels, in: 2014 IEEE Conference on Computational Intelligence and Games, 2014, pp. 1–8. doi:10.1109/CIG.2014.6932912.

[13] M. Carroll, R. Shah, M. K. Ho, T. Griffiths, S. Seshia, P. Abbeel, A. Dragan, On the utility of learning about humans for human-ai coordination, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems, volume 32, Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/f5b1b89d98b7286673128a5fb112cb9a-Paper.pdf.

[14] M. Charity, A. Khalifa, J. Togelius, Baba is y'all: Collaborative mixed-initiative level design, in: 2020 IEEE Conference on Games (CoG), 2020, pp. 542–549. doi:10.1109/CoG47356.2020.9231807.

[15] Mega Crit Games, Slay the spire, 2019.

[16] A. Teikari, Baba is you, 2019.

[17] Berlin Interpretation - RogueBasin, 2013. URL: https://www.roguebasin.com/index.php/Berlin_Interpretation.

[18] E. McMillen, F. Himsl, The Binding of Isaac, 2011.

[19] Supergiant Games, Hades, 2018.

[20] Nolla Games, Noita, 2019.

[21] Hopoo Games, Risk of Rain 2, 2019.

[22] M. Rosewater, Lenticular design, 2014. URL: https://magic.wizards.com/en/news/making-magic/lenticular-design-2014-12-15.

[23] M. Stephenson, J. Renz, Procedural generation of levels for angry birds style physics games, Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment 12 (2021) 225–231. URL: https://ojs.aaai.org/index.php/AIIDE/article/view/12871. doi:10.1609/aiide.v12i1.12871.

[24] R. Rodriguez Torrado, A. Khalifa, M. Cerny Green, N. Justesen, S. Risi, J. Togelius, Bootstrapping conditional gans for video game level generation, in: 2020 IEEE Conference on Games (CoG), 2020, pp. 41–48. doi:10.1109/CoG47356.2020.9231576.

[25] P. Taveekitworachai, F. Abdullah, M. F. Dewantoro, R. Thawonmas, J. Togelius, J. Renz, Chatgpt4pcg competition: Character-like level generation for science birds, in: 2023 IEEE Conference on Games (CoG), 2023, pp. 1–8. doi:10.1109/CoG57401.2023.10333206.

[26] M. Stephenson, J. Renz, X. Ge, L. Ferreira, J. Togelius, P. Zhang, The 2017 aibirds level generation competition, IEEE Transactions on Games 11 (2019) 275–284. doi:10.1109/TG.2018.2854896.

[27] C. Hu, Y. Zhao, Z. Wang, H. Du, J. Liu, Games for artificial intelligence research: A review and perspectives, 2024. URL: https://arxiv.org/abs/2304.13269. arXiv:2304.13269.

[28] S. Ontañón, N. A. Barriga, C. R. Silva, R. O. Moraes, L. H. S. Lelis, The first microrts artificial intelligence competition, AI Magazine 39 (2018) 75–83. URL: https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/2777. doi:10.1609/aimag.v39i1.2777.

[29] N. Heijne, S. Bakkes, Procedural zelda: a pcg environment for player experience research, in: Proceedings of the 12th International Conference on the Foundations of Digital Games, FDG '17, Association for Computing Machinery, New York, NY, USA, 2017. URL: https://doi.org/10.1145/3102071.3102091. doi:10.1145/3102071.3102091.

[30] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, J. Quan, S. Gaffney, S. Petersen, K. Simonyan, T. Schaul, H. van Hasselt, D. Silver, T. Lillicrap, K. Calderone, P. Keet, A. Brunasso, D. Lawrence, A. Ekermo, J. Repp, R. Tsing, Starcraft ii: A new challenge for reinforcement learning, 2017. URL: https://arxiv.org/abs/1708.04782. arXiv:1708.04782.

[31] B. Bateni, J. Whitehead, Language-driven play: Large language models as game-playing agents in slay the spire, in: Proceedings of the 19th International Conference on the Foundations of Digital Games, FDG '24, Association for Computing Machinery, New York, NY, USA, 2024. URL: https://doi.org/10.1145/3649921.3650013. doi:10.1145/3649921.3650013.

[32] W. Ling, E. Grefenstette, K. M. Hermann, T. Kočiský, A. Senior, F. Wang, P. Blunsom, Latent predictor networks for code generation, 2016. URL: https://arxiv.org/abs/1603.06744. arXiv:1603.06744.

[33] A. Santos, P. A. Santos, F. S. Melo, Monte carlo tree search experiments in hearthstone, in: 2017 IEEE Conference on Computational Intelligence and Games (CIG), 2017, pp. 272–279. doi:10.1109/CIG.2017.8080446.

[34] T. Chen, S. Guy, Chaos cards: Creating novel digital card games through grammatical content gen-

eration and meta-based card evaluation, Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment 16 (2020) 196–202. URL: https://ojs.aaai.org/index.php/AIIDE/article/view/7430. doi:10.1609/aiide.v16i1.7430.

[35] T. Chen, S. Guy, Gigl: A domain specific language for procedural content generation with grammatical representations, Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment 14 (2018) 9–16. URL: https://ojs.aaai.org/index.php/AIIDE/article/view/13025. doi:10.1609/aiide.v14i1.13025.