Hyperstate Space Graphs For Automated Game Analysis

Michael Cook
School of Electronic Engineering and Computer Science
Queen Mary University of London
London, United Kingdom
mike@gamesbyangelina.org

Azalea Raad
Max Planck Institute for Software Systems
Kaiserslautern, Germany
azalea@mpi-sws.org

***Abstract*—Automatically analysing games is an important challenge for automated game design, general game playing, and co-creative game design tools. However, understanding the nature of an unseen game is extremely difficult due to the lack of *a priori* design knowledge and heuristics. In this paper we formally define *hyperstate space graphs*, a compressed form of state space graphs which can be constructed without any prior design knowledge about a game. We show how hyperstate space graphs produce compact representations of games which closely relate to the heuristics designed by hand for search-based AI agents; we show how hyperstate space graphs also relate to modern ideas about game design; and we point towards future applications for hyperstates across game AI research.**

***Index Terms*—automated game design, general game playing, game analysis**

I. Introduction

Games are complex software that can be viewed in many different ways. Developers, critics, academics and players use different models to discuss and represent different parts of a game, to see something in a different light or focus on a particular game facet. In design and research a popular method for modelling the ways a game can play out is a *state space graph* (SSG), a directed graph in which the vertices represent *game states* and the edges represent *actions* taken by the player that transition the game from one state to another. SSGs are useful for a variety of purposes: designers may use them to model the structure of puzzles [6], and AI agents may construct and explore SSGs to evaluate the best next action [4].

Although SSGs are useful, most games are too large for exhaustive SSGs to be constructed. Many games have no fixed end point, which would result in a theoretically infinite SSG (e.g. *Civilisation*). Other games are simply very large – there are 361 initial moves in a game of *Go*, and after 50 moves the number of possible game states is in the order of 10^{127} [1]. The verbosity of videogame state spaces also means that important actions that have an impact on the game are dwarfed by a larger quantity of low-impact actions. For example, the large quantities of time spent running across the town in *Silent Hill 2* compared to the small but impactful time solving puzzles.

To manage the state space size, when constructing SSGs we often apply reduction or abstraction techniques. One option is to change the SSG representation. For example, in [6] a puzzle from *The Legend Of Zelda: Twilight Princess* is represented as an SSG, but many intermediate states, such as walking around rooms or the combat process, are either elided or combined into higher-level states. Another option, commonly used in the design of AI agents, is to limit the size of the SSG – many Chess agents are limited in how far ahead they may look to evaluate a move.

However, these abstraction processes for state space compression require *game-specific* expert knowledge: what is important and what is not, and which aspects of the game we wish to focus on. However, many AI systems are required to play, analyse or interact with games that they have not seen before, and in some cases have not been seen by any human before either. Automated game design systems must understand a game in order to be able to evaluate or change its design. Co-creative systems similarly must understand what a person has created in order to give feedback or respond creatively. General game playing agents must assess unseen games in order to devise or select an appropriate way of playing. For the general case of an unseen game design, no heuristics exist to help us compress or reduce the state space, or to guide us in evaluating the nature of this game.

In this paper we propose *hyperstate space graphs* (HSGs) as a general approach to compressing SSGs using the notion of *reversible actions*. We define a reversible action as any game action whose effects can be undone through a series of further game actions. This approach not only results in smaller graphs for many types of game, but the nature of the compression can help identify decisions which are important or more impactful, as well as providing metrics to measure the texture and distribution of choices in a game. Our early exploration shows promise as an analytical technique for automated game designers, but also has utility for other applications of AI to games, and may even be a useful analysis technique for people, too. This paper formally defines what reversibility means, how HSGs are constructed, how HSGs relate very closely to human-designed heuristics for search, and how they relate to modern theories about game design. We then close by discussing the future potential use and extension of this technique to various areas of game AI.

The remainder of the paper is organised as follows: in

The first author is supported by the Royal Academy of Engineering under the Research Fellowship scheme.

Hyperstate Space Graphs we formally define what it means for a game action to be reversible, and show how hyperstate spaces are constructed; in *Hyperstates As Heuristic Approximation* we show how hyperstate space graphs have a close relationship to the heuristics designed by human experts to solve games through search; in *Hyperstates As Game Design Theory* we show how hyperstate spaces concisely represent or relate to three modern ideas about game design; in *Related Work* we discuss existing work on adjacent problems; in *Discussion And Future Work* we discuss our plans for future work, and discuss specific applications to automated game design.

II. Hyperstate Space Graphs

A. State Space Graphs

A *game state* is a snapshot of information about a game at a particular instance in time. Depending on the granularity of the representation, a game state may be extremely detailed, even including information such as the contents of memory and pixel data in the case of a digital game. More often, people familiar with a game develop game-specific abstractions, limiting this information to a subset that is sufficient for a particular analysis.

For a given game (and, where appropriate, a given level) the *state space* is the set of all possible game states that are reachable from the game's initial starting state. Note that some researchers, such as [1], use the term 'state space' to refer to the set of all *describable* game states, including states which the game cannot reach through legal play (for example, a regular chess game with three kings in it). For our purposes, we use the narrower definition of *reachable from the initial game state*, since our analysis is focused on what possible ways there are to play the game.

Given a game's state space, we can construct a *state space graph* (SSG). An SSG, $\mathcal{G}{=}(G, a)$, is a directed graph, in which the vertices (G) are game states, and the edges (a) are *player actions*. Actions might be real-world actions (e.g. pressing the 'A' button) or in-game actions (e.g. jumping). Actions are represented as a labelled relation on game states: given states $g, g' \in G$, then $(g, l, g') \in a$ denotes that taking the action labelled l in state g, causes the game to transition to state g'. Figure 3a shows an example SSG with $G{=}\{g_1, \cdots, g_{11}\}$ and actions labelled $\{l_1, \cdots, l_{13}\}$.

Given an SSG $\mathcal{G}{=}(G, a)$, the *leaf* states denote those states without an outgoing action. For instance, in Figure 3a states g_3 and g_9 are leaf states. Each leaf state in an SSG is associated with a *win* or *fail* tag, denoting whether a game is won or lost. In Figure 3a, g_3 is tagged as a failure state, denoted by the square-shaped node, while g_9 is tagged as a success state, denoted by the diamond-shaped node. Note that not all leaf states may result in winning or losing: some games do not have success or failure built in, and there are many ways to *soft lock* games where no progress can be made. In such cases, leaves may be tagged with whichever concept makes sense – 'success' in games with no clear winning or losing condition, for example, and 'failure' in cases such as soft lock.

In its most basic form, an SSG shows every possible state the game can be in, and every possible action that can be taken in every state. However, due to their large size and complexity, state space graphs are rarely constructed in this way; instead, *game-specific* abstraction techniques or changes in representation are used to decrease this complexity. For example, in [6] a dungeon from *The Legend Of Zelda: Twilight Princess* is described using a graph where vertices correspond to key events in the dungeon's solution, and edges between them compress a series of actions into a single transition. This helps a designer or critic focus on the states and actions that are important (e.g. discovering a key item or fighting a mini-boss) and abstract away or de-emphasise states and actions that are less important (e.g. walking across an empty room to reach somewhere else). However, such techniques are highly *game-specific* and require expert knowledge about what can and cannot be abstracted away from a particular game's environment. To remedy this, we propose the concept of *reversible game actions* as a general abstraction technique that requires no game-specific knowledge or expertise.

B. Reversible Actions

An action is *immediately reversible* iff its effects can be undone (reversed) by another action. Figure 1 shows an example of an immediately reversible action in the game *Sokoban*, a turn-based, grid-based puzzle game. The player presses the down arrow, which moves the player character down one tile. Pressing the up arrow *reverses* the action by returning the player to their original location.

Put formally, given $\mathcal{G}{=}(G, a)$, an action $(g, l, g') \in a$ is *immediately reversible* iff there exists an action labelled l' which transforms g' into a game state g'' (i.e. $(g', l', g'') \in a$), such that $g \equiv g''$, where $\equiv$ denotes state *equivalence*. State g is equivalent to g'' ($g \equiv g''$) if *all* data that can impact the game's execution has the same value in g and in g''. This means, for example, that the position of the player's character is compared, but so is any internal state the character has, or any external variables such as the player's score or narrative flags. As we discuss in future work, it is possible to use a weaker definition of game state equivalence.

Note that $g \equiv g''$ holds if $g = g''$, i.e. when the game is returned to the *exact same* state. For instance, in the *Sokoban* example of Figure 1, pressing the up arrow reverses the action by returning the player to the *same* original state. In the SSG example of Figure 3a, the action $g_1 \xrightarrow{l_{10}} g_2$ is reversible, since the action $g_2 \xrightarrow{l_8} g_1$ reverses its effect.

Some actions may not be *immediately* reversible, but rather *eventually* reversible: they can be reversed through a *sequence* of actions. Figure 2 shows an example of eventual reversibility in *Sokoban*, where a crate can be pushed back into its original position and the player returned to the starting location with a longer sequence of moves. Put formally, given an SSG $\mathcal{G}{=}(G, a)$, an action $(g, l, g') \in a$ is *eventually reversible* iff there is a sequence of actions with labels $l_1, ...l_n$ which, when applied successively to g', yield a state g'' such that $g \equiv g''$. Note that the actions labelled $l_1, ...l_n$ are not

Fig. 1: An *immediately reversible* action in the game *Sokoban*. Moving down (left) only modifies the player position, which is trivially reversible by moving up (right).

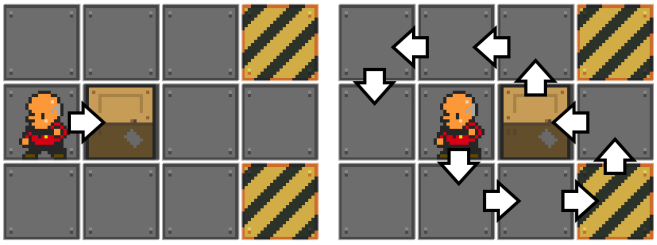


Fig. 2: An *eventually reversible* action in *Sokoban*. Pushing the crate (left) is not immediately reversible, but it can eventually be pushed back into its original position before the player returns to their original location (right).

necessarily immediately reversible themselves, but they are all eventually reversible. In the SSG example of Figure 3a, the action $g_4 \xrightarrow{l_4} g_5$ is eventually reversible since the sequence $g_5 \xrightarrow{l_5} g_6 \xrightarrow{l_6} g_7 \xrightarrow{l_7} g_4$ reverses its effect. Observe that the actions labelled l_5, l_6 and l_7 are also eventually reversible.

We refer to immediately reversible and eventually reversible actions collectively as reversible actions. Reversible actions capture an important feature of games, namely the distinction between *tentative* and *permanent* actions. Tentative actions do not commit the player to a particular decision, and can be reversed somehow. For example, moving the player around the game space in Sokoban, exploring a maze in a puzzle game, rearranging your inventory in a roleplaying game, or examining an item in an adventure game are all examples of tentative actions. By contrast, permanent actions change the game in a way the player cannot completely undo. For example, destroying tiles in a Match-3 game, giving a gift in a dating game, or building a school in a city management game are all examples of permanent actions. While some of these actions may be reversible in part (e.g. bulldozing the school), other side effects of the decision cannot be reversed (e.g. resources spent to build the school, a change in citizen happiness, or the time that passed during which the school existed). We use this insight to *abstract* an SSG to a *hyperstate space graph* (HSG) by merging together those states that are connected by reversible actions. That is, HSGs abstract (hide) tentative actions and only contain permanent actions, reducing the state space size significantly in most cases.

C. Hyperstate Space Graphs

A hyperstate space graphs (HSG) is built from an SSG by *merging together* those states that are connected by reversible actions. For instance, the HSG associated with the SSG in Figure 3a is given in Figure 3b. Note that as states g_1, g_2 are connected by a reversible action, they are merged together into state $\{g_1, g_2\}$ in the corresponding HSG. Similarly for states g_{10} and g_{11}. Analogously, since states g_4, g_5, g_6, g_7 are connected by an eventually reversible action, they are merged together since they are merged together into state $\{g_4, g_5, g_6, g_7\}$ in the HSG. The remaining states are left unchanged as they do not have reversible actions. We now formalise the definition of HSGs.

Given an SSG $\mathcal{G}{=}(G, a)$, we write a^* to denote the *reflexive, transitive closure* of a. That is, whenever $(g,, g') \in a$ and $(g',, g'') \in a$, then $(g,, g') \in a^*$ (transitivity), and $(g,, g), (g',, g'), (g'',, g'') \in a^*$ (reflexivity). Given a game state $g \in G$, we write $a^*(g)$ to define the set of states in the reflexive transitive closure of g:

$$a^*(g) \stackrel{\text{def}}{=} \{g' | (g, -, g') \in a^*\}$$

We define the *reversible closure* of a state $g \in G$, denoted $\mathcal{G}.rc(g)$, as follows:

$$\mathcal{G}.rc(g) \stackrel{\text{def}}{=} \{g' |\ g' \in a^*(g) \wedge g \in a^*(g')\}$$

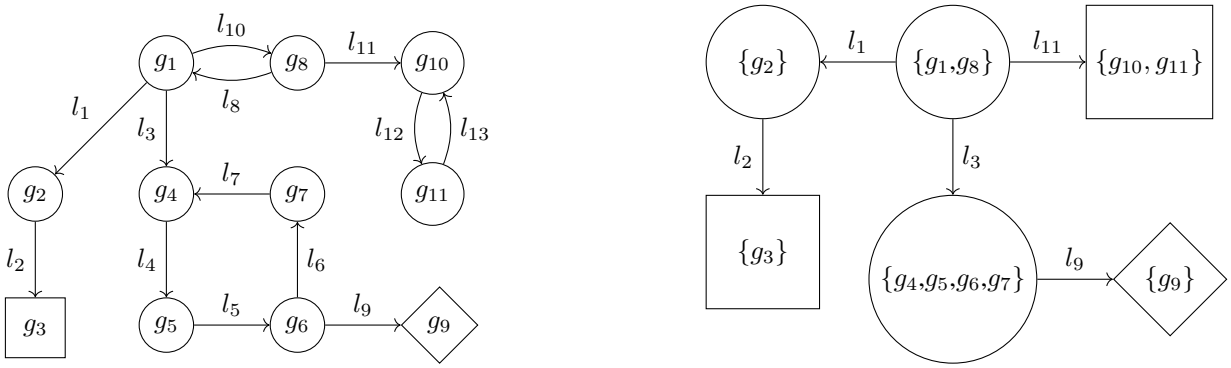
That is, a state g' is in the reversible closure of g if g' is in the reflexive transitive closure of g and vice versa; i.e. they are mutually reachable from one another. Given this definition, we can now define the structure of a HSG.

Given an SSG $\mathcal{G}{=}(G, a)$, its associated HSG is given by $\mathcal{H}{=}(H, c)$, where H is a set of *hyperstates* and c is a relation on H describing action transitions on hyperstates such that:

1) Each hyperstate $hs \in H$ is a *set* of game states in G (e.g. the hyperstate $\{g_1, g_2\}$ in Figure 3b contains the states g_1, g_2 of the original SSG in Figure 3a).
2) Each state $g{\in}G$ appears in *exactly one* hyperstate $hs{\in}H$.
3) Each hyperstate contains states that are mutually reachable via reversible actions; for instance, g_1 and g_2 in Figure 3a are mutually reachable via reversible actions labelled l_8 and l_{10} and thus form the hyperstate $\{g_1, g_2\}$ in Figure 3b. Formally, for all hyperstates $hs \in H$ and all states $g_1, g_2 \in G$, if $g_1, g_2 \in hs$, then g_1 and g_2 are in the reversible closure of each other, and vice versa:

$$\forall hs \in H. \forall g_1, g_2 \in G.$$
$$g_1, g_2 \in hs \iff g_1 \in \mathcal{G}.rc(g_2) \wedge g_2 \in \mathcal{G}.rc(g_1)$$

4) The HSG actions are simply lifted from SSG actions: if $(hs, l, hs') \in c$, then there exists $g \in hs$ and $g' \in hs'$ such that $(g, l, g') \in a$, and vice versa; for example, we have $\{g_1, g_8\} \xrightarrow{l_1} \{g_2\}$ in Figure 3b because of $g_1 \xrightarrow{l_1} g_2$ in Figure 3a. Moreover, we *exclude self-loops*, i.e. actions between the *same* hyperstate; for example, although we have $g_1 \xrightarrow{l_{10}} g_8$ in Figure 3a, we exclude $\{g_1, g_8\} \xrightarrow{l_1} \{g_1, g_8\}$ from Figure 3b.



(a) A state space graph

(b) The hyperstate space graph constructed from Fig. 3a

Fig. 3: An SSG (left) and its corresponding HSG (right); square nodes denote failure and diamond nodes denote success.

Note that while relation a in the SSG represents *all* game actions that transition between states, relation c in the corresponding HSG represents *only non-reversible* actions. We refer to such actions as *critical actions*, hence the notation c.

a) Automating HSG Construction: We have developed an algorithm that automates the HSG construction process. That is, given an SSG it is always possible to compute its associated HSG such that it satisfies the above conditions. Given an SSG $\mathcal{G}$, our algorithm (i) identifies the hyperstates of the associated HSG; and (ii) adds the appropriate edges between these hyperstates. For step (i), we use a cycle detection mechanism to identify mutually reachable states in $\mathcal{G}$, and subsequently merge the nodes of each cycle into a single hyperstate. For step (ii), we iterate over the $\mathcal{G}$ edges and for each (g, l, g') edge, we add the edge (hs, l, hs') to our HSG, when $g \in hs$, $g' \in hs'$ and $hs \neq hs'$.

1) Tagging Hyperstates: As with SSG leaf states, each HSG leaf state is associated with a *win* or *fail* tag. A leaf hyperstate is tagged as a *win hyperstate* if it contains *exactly one* win state; e.g. $\{g_9\}$ in Figure 3b is tagged as a win hyperstate since g_9 is a win state in Figure 3a. Analogously, a leaf hyperstate is tagged as a *fail hyperstate* if it contain *no* win states; e.g. both $\{g_3\}$ and $\{g_{10}, g_{11}\}$ are tagged as fail hyperstates as they contain no win states of Figure 3a.

Once all leaf hyperstates are tagged, we next tag non-leaf hyperstates as either *eventually winnable* or *implicitly failed*. A non-leaf hyperstate hs is tagged eventually winnable if it can reach at least one win leaf hyperstate. A non-leaf hyperstate is tagged *implicitly failed* otherwise. For instance, in Figure 3b the non-leaf hyperstates $\{g_1, g_8\}$ and $\{g_4, g_5, g_6, g_7\}$ are both tagged as eventually winnable as they can reach the win leaf hyperstate $\{g_9\}$, while the non-leaf hyperstate $\{g_1, g_8\}$ is tagged as implicitly failed as it cannot reach $\{g_9\}$. Implicitly failed hyperstates are functionally the same as fail hyperstates, as there is no way for the game to resolve except in a loss, even though the player can still interact with the game.

For those familiar with modal logic, such tagged hyperstates can be defined using reachability modalities when interpreting the HSG as a Kripke graph. In particular, when W describes the win (leaf) hyperstates, eventually winnable (non-leaf) hyperstates are described by $\Diamond W$ (read 'eventually W'), while implicitly failed hyperstates are described by $\neg\Diamond W$, or equivalently by $\Box\neg W$ (read 'always not W').

III. Hyperstates As Heuristic Approximation

As discussed in §I, when we develop abstract representations for games we often use game-specific heuristics we have discovered, or our own intuition, in order to find efficient models that focus on the most important or relevant aspects. This is also true of AI applications to specific games. For example, [8] describes a reduction of the Sokoban game space such that it removes actions which are uninteresting for the solver, making the search problem simpler. The authors note:

> *Two states are the same, if the box positions are identical and the player can move from the position it occupied in the one state to the position it occupies in the other state without moving a box.*

This intuition about Sokoban's state space and the action relation on it – namely, that movement is not interesting unless it results in boxes moving – compresses Sokoban's game state space drastically, allowing the search-based agent to focus on decisions which are important and have meaning. This heuristic for Sokoban is analogous to the notion of non-eventual reversibility; that is, reversibility within a single move. Movement in Sokoban is immediately reversible, while moving boxes is not. As a result, the AI solver's state space only involves moves which affect boxes.

Our HSG representation actually takes this a step further, because eventual reversibility also means that box movements which can be reversed (such as the one shown in Figure 2) are also condensed into a single state. The reason that [8] does not take this approach is likely because it is hard to concisely describe which box moves are reversible and which are not. Our hyperstate space graph construction obtains this knowledge through a deep exploration of the state space, whereas a heuristic is designed to be a general guideline. Nevertheless, there is a close correspondence between our notion of reversibility and the authors' intuition about how to collapse Sokoban's state space.

[9] describes the design of a framework for playing games using Monte-Carlo Tree Search (MCTS - see [5]). In one of their case studies they describe a game similar to Tetris, where falling blocks represent parts of monsters that must be connected to one another. The authors note the following when discussing the action space for their MCTS agent:

> *Thus, at any point, she has four to six possible keyboard-level actions: rotate, move left, move right, quick-land, cycle highlighter, collect monster. Some of those actions could be repeated indefinitely without affecting the game state, meaninglessly expanding the scope of the search for Monster Carlo to perform. To avoid this, we use micro-decisions to model only those choices.*

In this example, the 'micro-decisions' are chosen by the designer of the system, using their specialised knowledge about the game's decision space. Once again, we can see that actions which 'could be repeated indefinitely without affecting the game state' corresponds to our notion of reversibility, and the solver has been explicitly designed to ignore such distinctions. Our HSG representation would create a similar reduction if applied to this game.

We believe that these examples support the idea that HSGs represent an intuitive compression of game state spaces that is closely related to how the developers of AI agents compress state spaces to make them amenable to search-based agent algorithms. Even though our system ideally requires an exhaustive exploration of a specific search space in order to determine hyperstates, we believe that once a hyperstate space graph is generated it may be possible to infer a more general heuristic from it. For example, by constructing the hyperstate space graph of a small set of Sokoban levels, a more general heuristic that focuses only on box movement might be automatically derivable based on the kinds of states which were compressed. We are currently pursuing this as an extension to this work.

IV. Hyperstates As Game Design Theory

The main motivation of this work is to provide a lens for automated game analysis for software working in domains such as automated game design or general game playing, in a way that required no game-specific knowledge. Additionally, however, we have found that HSGs and the notion of action reversibility align well with several informal ideas about game design. In this section we describe three such ideas and show how they relate to or are expressible with HSGs. This suggests that HSGs may have a use as a general formalism for expressing certain game design ideas, as well as supporting our argument that HSGs capture a common and abstract way of thinking about games.

A. Strategic Headroom

In [13], Smith informally defines the notion of *strategic headroom*, with respect to the roguelike genre of games. Smith is one of the active developers of NetHack, one of the longest-running and best-known roguelike projects in the world. Smith's essay defines what strategic headroom means for a game, and how roguelikes can be broadly classified according to how much headroom they wish to afford the player. Strategic headroom is a measure of how much freedom the player has to make suboptimal choices and still complete the game. Giving an example of a simple binary choice in a game, Smith explains:

> *There are two possible extremes for the scale here. One is "no matter who you are or which choice you make, it won't affect your odds of winning the game"... The other extreme is "one of these choices will guarantee you victory; the other will guarantee you are defeated".*

These extreme cases are examples of high and low headroom respectively. The former has infinite headroom, while the latter has zero headroom. Smith posits that neither approach is intrinsically good or bad: different players will seek out different kinds of headroom, and different game designs benefit from a different emphasis on headroom.

Although Smith only analyses roguelikes, one can think of headroom more generally as a way of capturing a particular property of the game's decision space. Given an SSG $\mathcal{G}{=}(G, a)$ and a state $g \in G$, the headroom of g is directly related to what proportion of states in $a^*(g)$ (the states reachable from g) are winnable. If g has maximum or infinite headroom, then the decision made in g does not have any impact on the outcome of the game. In this extreme case, this means not only that every state in $a^*(g)$ is winnable, but that it is *perfectly* winnable: all actions in $a^*(g)$ lead to a win state.

Similarly, we can consider an equally extreme state of low or zero headroom, where there is only one viable action from a given state, and all other actions lead to failure. In HSG terms, this means that there is only one action that leads from g to a winnable state; all other actions lead to implicit or actual failure states. Unlike the maximum headroom example, minimal headroom does not imply anything about states more than one action away from g.

More generally, the headroom of a game state g can be thought of as a ratio of eventually winnable to implicitly failed states in the closure of g (via $a^*(g)$); that is, all the states accessible from g through any valid sequence of actions. Higher headroom states will have a larger proportion of winnable states in $a^*(g)$, particularly in states only one action away from g.

There are certain nuances of Smith's original definition which would best be expressed by extending our definition of HSGs. In particular, roguelikes rely heavily on stochastic systems, which means the outcome of many actions in roguelikes is non-deterministic. This means that for a given state g and an action labelled l, there may be more than one g' such that $(g, l, g') \in a$. This is a specialised type of HSG which we plan to investigate further in the future. However, the notion of headroom is useful in both deterministic and non-deterministic scenarios.

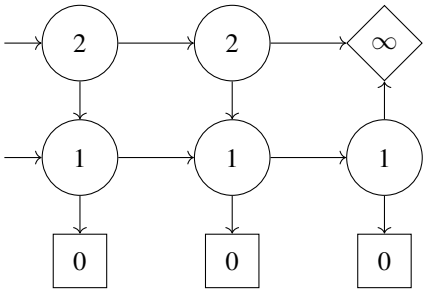


Fig. 4: A partial HSG. Diamonds denote success states and squares denote failure states; each state is labelled with the shortest distance (number of actions) to a failure state.

B. Failure Spectra

In [7], the author introduces the notion of a 'failure spectrum', describing the staggered nature of failure in some games, whereby an action can make a situation worse without outright ending the game. Francis writes:

> *When you can fail at something but still carry on playing, I call the range of states between perfect success and total failure a failure spectrum: there's a spectrum of possible outcomes, and screw-ups can move you towards the failure end and recoveries can (sometimes) move you back up towards success.*

Many kinds of game exhibit failure spectra. Francis identifies stealth games as a good example of the concept: failures in such games typically expose the player more and more, from their initial hidden state into open combat or fleeing from danger, as enemies become increasingly aware of their presence. In many such games, this is recoverable through hiding, covering up evidence, or acting in a particular way. However, this recoverability is often imperfect – for example, in *Invisible Inc*, an alerted guard will search for the player even after they lose track of them.

A move down the failure spectrum towards total failure can be thought of as a transition towards states which are closer to implicit or actual failure in the hyperstate graph. Figure 4 shows a partial hyperstate graph for a game, where states are labelled with their *loss distance*, or ld, denoting the shortest distance to a loss state. An ideal path through the corresponding HSG always chooses the action which leads to the state with the highest ld. The greater the distance between the ideal path and the nearest loss state, the broader the failure spectrum is for the game.

C. Depth

In [11], the authors attempt to formalise a quality which approximates the phenomenon of 'depth' in strategic games. In particular, they suggest that depth may be related to the space of possible strategies for playing the game, and how those strategies are ordered, with a deep game offering a wide spectrum of strategies, which increase in both computational complexity and performance. Lantz et al also consider other metrics that might correspond to depth, including game state space, about which they say:

> *However, state space by itself cannot explain the qualities of depth that we are looking to isolate. To see why, imagine any existing game and then add a new rule that allows either player, after their turn, to flip a token from side A to side B. This immediately doubles the size of the game's state space without affecting its depth in the slightest.*

Although, as they argue, flipping a token doubles the size of an SSG, it leaves the size of the corresponding HSG unchanged; since flipping a token is trivially reversible, any two game states connected by flipping a token will be merged into the same hyperstate, resulting in no change in the size of the hyperstate space. In fact, flipping a token has no impact on game execution other than affecting how the token can be flipped in the future, and so could be excluded from the equivalence relation altogether.

While new pathological cases could be invented which cause problems for hyperstate space graphs, we find it interesting that our representation circumvents the natural counterexample proposed by the authors. Although we do not believe that hyperstates provide a direct correlate to depth, there does seem to be some link between the way in which hyperstates reduce state space graphs, and the author's dissatisfaction with ordinary state space as a metric for depth. We hope to explore this avenue in the future, as we investigate the utility of various hyperstate-based metrics for measuring qualities about a game.

V. Related Work

While we are not aware of any work specifically dealing with general representations for exhaustive game analysis, there is a large body of work that deals with overcoming the challenges of navigating a game's state space. The history of graph search algorithms, especially as the area developed techniques for game agent AI, is aimed at this type of problem. For example, Alpha-Beta search explores the game state space by pruning subtrees that are known to be less optimal than an already discovered alternative move [10]. This analysis is designed to make a single choice for an agent's next action, which means it is not interested in understanding the overall shape of all choices on offer. This is true of many search algorithms, such as Monte Carlo Tree Search (MCTS), which combines precise action selection with random exploration to balance its search through a state space [5].

Another approach to the problem of analysing unseen games is to develop a set of known analytical approaches for existing games, and then cluster them such that new games can be matched to the nearest cluster. [2] applies such a clustering approach on several VGDL games (see [12]), considering features such as whether the player can die, what sprites are on the screen, and how many interactions the game contains. These features help assign a new game to a cluster, which has an associated player control algorithm which is selected for use. This is an innovative approach to general game playing (GGP) that is especially suited to many GGP benchmarks

where games are expected to have similar properties to classic games. Our approach differs in that we do not require prior clustering, or for the unseen games to be similar to any existing game. This is particularly important for domains such as automated game design or working co-creatively with people, as in both cases, it is desirable to work with games that may be highly innovative or break convention.

In [3] the authors describe a system for evolving new game designs in the genre of two-player adversarial abstract games. Part of their system evaluates candidate game designs to find what they believe to be higher quality games and filter out lower quality ones. The system is provided with several human-developed heuristics, but they are intended to be fairly broad and applicable to the genre of two-player adversarial games, rather than specific to a particular game. Many of these metrics are based on an analysis of game playouts, for example a 'drama' measurement looks at how many times the advantage swings between one player and another. This represents a design-first approach to dealing with unseen games, where the authors' significant knowledge of the domain leads to specific (but genre-wide) heuristics about desirable game properties.

Brown and Maire's approach, like many automated game design approaches, work well in that it leverages specific domain knowledge from the authors to precisely evaluate certain features of games in a known space. Our work differs here in several ways. First, Brown and Maire's approach uses the authors' personal experience with games of this type in order to develop specific heuristics about what makes such a game good. The authors are clear that they are not claiming their heuristics to be objectively true or universally applicable, rather that they are expressions of their own experience in the field. Our approach differs here in that it requires no prior game-specific knowledge or experience. Moreover, we avoid using specific claims about what we believe a *good* game to be, and instead aim to provide a representation that reveals interesting structures and metrics to distinguish between *types* of game, challenge or experience. Rather than say 'this game is good' or 'this game is bad', we intend for hyperstates to be a way to look at the DNA of a game experience and see what properties it has, what texture the landscape of decisions has, so that automated systems can make more interesting decisions in the absence of specific human-originated design knowledge.

In addition to this, Browne and Maire's work is focused specifically on two-player adversarial games, and their metrics make frequent reference to which player is likely to win, or how often a game ends in a draw. Our approach is broader, and applicable to any type of game (with extensions – see *Discussion and Future Work* below). This makes sense, since their system was designed specifically to create this type of game, whereas we have aimed for a more general scope in order to cater to a wider spectrum of automated game design applications. Finally, due to their use of search-based game playing agents, their system primarily explores parts of the state space that lead to victory as efficiently as possible. By contrast, we are interested in surveying the fuller extent of the game's solution and failure space, including suboptimal play and esoteric failure states.

VI. Discussion And Future Work

A. Applications To AI And Games

Evaluating partial or complete game designs is a challenging problem within the field of automated game design as well as co-creative design. For procedural content generation tasks such as level generation for platforming games, we can embed existing design knowledge and heuristics about the target domain in order to improve the generator's assessment of artefacts it generates. By contrast, in the case of game design few general heuristics exist, and even personal guidelines or rules of thumb are usually expressed in highly subjective terms (such as Sid Meier's definition of a game as 'a series of interesting choices'). Both automated game design systems and co-creative game design tools cannot use such heuristics (since we cannot define 'interesting' or 'fun' or similar ideas mathematically), and must thus look for other ways to evaluate and analyse game designs.

HSGs can help automated game designers in several different ways. First and foremost, HSGs greatly compress state space graphs for many games, which makes the act of analysis more efficient. In our pilot experiments using the automated game designer ANGELINA, we have also noted that low-quality games tend to be highly compressible, because their rulesets tend to be unfocused and their level designs sparsely populated. HSGs can help identify these games and more conclusively set them apart.

The second benefit is that they provide a means for comparing different games. Comparing games with one another using classic SSGs is difficult, because the structure of an SSG does not reveal much about what the game is like to play. SSG features such as the branching factor or the number of win or loss states can be misleading, and in the absence of heuristics that clarify this, comparing two unknown games is unreliable. As we showed earlier when discussing heuristic search, HSGs compress less important actions in a game and thus reduce the state space of games down in similar ways. While this does not make it possible to say if a game is 'better' than another, it does provide a better way to inspect what the decision space of two games looks like.

A final benefit is that HSGs provide metrics which can be used either for comparative purposes or to evaluate single games. Measurements such as the length or quantity of winning paths, the percentage of game states compressed in the creation of the HSG, the number and nature of reversible actions from a given path, all provide additional insights into the kinds of choices players make while playing a game. We are currently conducting research into what uses different metrics may have, with an initial focus on distinguishing different types of challenge in puzzle games.

These benefits should carry over to research in GGP, where similar problems related to playing unseen games often arise. For example, in [2] the authors present a system which analyses a game and then selects from a catalogue of algorithms

based on what it thinks will best suit of game it is faced with. This analysis uses clustering to analyse known games based on features such as whether the player can die, or what proportion of game objects are related to a win condition. We can imagine a similar GGP approach that incorporates metrics about the frequency of choices, the percentage of choices which are critical, and the number of states that can be compressed in an HSG. For example, a high compression rate suggests a low number of critical decisions and a fairly sparse decision space, which might benefit longer rollouts and less search.

B. Non-Exhaustive Hyperstate Space Graphs

In some cases, calculating a full HSG is impractical, while for other cases it may simply be unnecessary. In particular there are two cases where a different approach might be preferable: infinite or effectively infinite state spaces, and extremely granular state equivalence. In this section we propose two extensions to HSG construction, the properties of which we intend to explore more in the future.

1) Depth-Limited Hyperstate Space Graphs: As discussed in the introduction, some games either have no ending (such as some games of *Civilisation*) or have such a large state space that it is effectively infinite. In such cases, we can use a depth limiting technique similar to that used by search-based game-playing agents. However, unlike a traditional SSG where depth-limiting has no effect on the structure and content of the graph (other than to curtail its size), a depth-limited HSG may be different in structure and content.

Suppose we limit the depth of our initial SSG to paths of length 10. Now suppose that an action a_j taken at the initial state g_i is *eventually reversible* using a sequence of 11 actions. Due to the limited search depth, this sequence is never discovered, and so a_j is marked as non-reversible and the states it connects may not be combined into a single hyperstate. This is a correct representation of the game's decision space under the 10-move depth limit, but nevertheless differs from the structure of the exhaustive HSG.

Rather than limiting the depth explored in the SSG, another approach is to limit the depth of the HSG instead. In this approach the SSG would be generated as normal, and the HSG would be iteratively constructed as the SSG is generated, stopping when the HSG reaches a certain depth. This approach would create more consistent HSGs when comparing games or levels, however the execution time and space constraint of the approach is less consistent - highly compressible games would take far longer to generate than less compressible games, and perfectly compressible games (that is, games which compress to a single hyperstate) would be generated in their entirety.

We intend to investigate the utility of depth-limited HSGs in future work, and test out how the analysis of games is affected by non-exhaustive HSGs. Depth limits would make HSGs more efficient for realtime use, and work better for time-limited applications such as game-playing agents, as it would allow a partial HSG to be compiled at any time.

2) Weak Equivalence Hyperstate Space Graphs: In our earlier definition of reversibility we defined a game state equivalence relation $\equiv$, such that in the most complete definition of reversibility, equivalence should include *any* data or information that could affect the execution of the game. For many genres of game, e.g. point-and-click adventures, interactive fiction or turn-based puzzle games, this data is likely to be of a manageable size. However, for some types of game it may be desirable to abstract away certain details from the state equivalence relation in order to focus on elements of the game that are more important.

For example, *Super Mario Bros.* has a timer which counts down during play, and has several effects on the execution of the game (e.g. causing a game over). For some applications, the timer is an important element of a game state and it would be necessary to include it when comparing states to one another, making any two states with different timer values different from one another under $\equiv$. However, in some cases one can weaken the notion of $\equiv$ to abstract away the timer from the equivalence relation, leading to a smaller HSG. That is, under this weaker notion of $\equiv$, two states are considered equivalent if they have the same data *up to and excluding timer values*. This would not contain information about losing the game due to running out of time, or getting a higher score by finishing levels more quickly, but it would be able to compress a large number of otherwise identical states into one another, resulting in a more compact HSG.

Earlier in this paper we motivated this work in part by stating that heuristics for state space compression are too nuanced to be discovered automatically by software. However, identifying game elements that have a large impact on compressibility but a low impact on execution is quite possible when using HSGs, and we intend to experiment with the automatic discovery of these features in future work. This would potentially allow game analysis systems to generate smaller HSGs with minimal loss of information.

VII. Conclusions

We introduced the notion of a *reversible actions* in a game, and showed that this allows us to compress the state space of certain games into a *hyperstate space*. HSGs provide new metrics with which to quantify the nature and distribution of decisions made in a game, even if the game has not been seen before, which makes it highly applicable to certain kinds of automated game design, co-creative game design, and general game playing. In addition, we showed how this affords precise ways for expressing certain ideas about game design, and illustrated this by showing how reversibility and hyperstates relate to contemporary ideas about elements of game design. Finally, we discussed the future of this work, including extensions to the hyperstate representation and its application to level generation.

Hyperstates were conceived as a way to gain a more expressive set of metrics for automated game designers to use to analyse and decide about games and game content. However, we believe that this formalism has wider applications beyond automated game design, and look forward to developing it further as a representation of game space.

REFERENCES

[1] Louis Victor Allis. *Searching for Solutions in Games and Artificial Intelligence*. PhD thesis, Maastricht University, 1994.

[2] Philip Bontrager, Ahmed Khalifa, Andre Mendes, and Julian Togelius. Matching games and algorithms for general video game playing. In *Proceedings of the Twelfth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2016.

[3] C. Browne and F. Maire. Evolutionary game design. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(1):1–16, 2010.

[4] Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, Peter I. Cowling, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions On Computational Intelligence And AI In Games*, 2012.

[5] Hyeong Soo Chang, Michael C. Fu, Jiaqiao Hu, and Steven I. Marcus. An adaptive sampling algorithm for solving markov decision processes. *Operations Research*, 53(1):126–139, 2005.

[6] Joris Dormans. Adventures in level design: Generating missions and spaces for action adventure games. In *Proceedings of the Workshop on Procedural Content Generation in Games*, 2010.

[7] Tom Francis. What works and why: Invisible inc, 2014. https://tinyurl.com/failurespectra.

[8] Nils Froleyks. *Using an Algorithm Portfolio to Solve Sokoban*. PhD thesis, Karlsruher Intitut fur Technologie, 2016.

[9] Oleksandra Keehl and Adam Smith. Monster carlo: an mcts-based framework for machine playtesting unity games. In *Proceedings of the IEEE Conference on Computational Intelligence in Games*, 2018.

[10] D. E. Knuth and R. W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293–326, 1975.

[11] Frank Lantz, Aaron Isaksen, Alexander Jaffe, Andy Nealen, and Julian Togelius. Depth in strategic games. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*. AAAI, 1 2017.

[12] T. Schaul. A video game description language for model-based or interactive learning. In *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, 2013.

[13] Alex Smith. Strategy headroom in roguelikes. NetHack 4 Development Blog, 2014. https://tinyurl.com/stratheadroom.