Rinascimento: Playing Splendor-like Games with Event-value Functions

Ivan Bravi*, Simon Lucas, *Senior Member, IEEE*

***Abstract*—In the realm of games research, Artificial General Intelligence algorithms often use score as main reward signal for learning or playing actions. However this has shown its limitations in scenarios where the rewards are very rare or absent until the end of the game. The problem is even more severe when the computational budget available is limited. This paper proposes a new approach based on event logging: the game state triggers an event every time one of its features changes. These events are processed by an Event-value Function (EF) that assigns a value to a single action or a sequence. Experiments show that this approach can mitigate the problem of scarce rewards and improve the AI performance compared to both point-based heuristics and State-value Functions (SF). Furthermore this represents a step forward in a finer control of the strategy adopted by the artificial agent, by describing a much richer and controllable behavioural space through EFs. Tuned EFs are able to neatly synthesise the relevance of the events in the game. Agents using an EF are also more robust when playing games with several opponents.**

***Index Terms*—Artificial Intelligence, Computational Intelligence, Games, Decision Making, Statistical Forward Planning**

I. Introduction

GAMES offer a multitude of possible applications of AI algorithms: game-playing, procedural content generation, player modelling, analytics and more. Game-playing algorithms can either plan their actions using a model of the environment to simulate actions (e.g. Monte Carlo Tree Search (MCTS) [1] and Rolling Horizon Evolutionary Algorithm (RHEA) [2]), or learn a policy or value function through direct interaction with the environment (e.g. Deep Reinforcement Learning (DRL) [3]). These approaches can also be combined to great effect e.g. AlphaZero [4].

Whether learning or planning, the process usually exploits the presence of an in-game score as a measure of good gameplay, a reward signal. Such approaches are commonly used in general game-playing frameworks such as the Arcade Learning Environment (ALE) [5] and the General Video Game AI (GVGAI) [6]. Unfortunately in many games these rewards can be very rare or absent until the end of the game, making the use of planning or learning particularly expensive.

In this paper we explore a new idea: to directly learn the value of events. The intuition behind this, is that events are key to any game, and that it may be easier to learn their value directly, than to learn a State-value Function (SF) that typically reflects the combined effects of many different events that occurred at different times. The counter-argument is that the game state represents a distillation of all those events, and so contains all that matters. However we argue that events have the unique ability of outlining the direction that the gameplay is taking which a single representation of the game state can not have. Furthermore, even if the information content was theoretically equivalent, learning directly the values of events in the shape of an Event-value Function (EF) could still be advantageous as they provide a more compact and explicit representation of the game dynamics. This representation can be used by game designers to analyse the game and improve or fix unwanted dynamics of it.

Another important point that we make in this paper, and rarely approached in Game AI, is guaranteeing diversity in the AI's game-playing style (later in the paper we will use *game-playing style* and *behaviour* interchangeably). For Statistical Forward Planning (SFP) methods, those using a forward model, we can achieve some degree of diversity by choosing different hyperparameter settings for an agent (e.g. how far ahead it is planning), but we are still bound to a winning or scoring signal. More specifically, we don't have much control on what happens between the signals, e.g. in Super Mario Bros.™ what Mario does between killing one Koopa and the next. A different story is for DRL where the black-box nature of the algorithms makes characterising their ability to express different styles not as straightforward.

Analysing events may promote behavioural diversity as the AI decision making is based on a set of semantically-rich features that unlock a deeper control of playing style than a non-tunable scalar signal i.e. the score. Behaviour expressivity is crucial aspect of automatic playtesting, we want an algorithm/player model that can express enough strategies so to completely cover the space of strategies in the game tested. This becomes even more important in a multiplayer game, in fact, opponents will influence the optimal strategy required. For this reason, in this context any improved performance-oriented proficiency is welcomed but won't be a make or break factor.

In this paper we compare several value functions: score, Event-value and State-value functions. These functions, used by a base SFP agent that provides them the results of action simulations, will produce a value synthesis of the actions simulated. It's already clear how EFs and SFs focus on two different aspect of the game, respectively its dynamic and its static aspects. Event-logging may seem to require a conspicuous engineering overhead, on the contrary, it can be swiftly implemented with an appropriate logging library directly in the game source which is available in most game AI applications. In fact, logging is already implemented in

I. Bravi and S. Lucas are with the Departmentof Electrical Engineering and Computer Engineering (EECS), Queen MaryUniversity of London, London E1 4NS

* I. Bravi is corresponding author: i.bravi@qmul.ac.uk

many video games for telemetry and analytics purposes.

To summarise, we present a novel approach to reshaping the reward landscape based on *events*: rewards are just the culmination of a series of events triggered by the players and the environment. Monitoring events is a more fine-grained approach that can fill in the gaps between sparse rewards creating a gradient to follow. Events are clearly game-specific, however the methodology making use of them can be still regarded as generic.

This paper is a follow-up work to [7] where we expand the comparison ground to also state-value functions thus showing further proof of strength for Event-value Functions.

Section II discusses the state of the art where this research takes place, Section III provides an overview of the value functions compared with subsection III-B giving a formal definition of *event-value function* and its implementation. Section IV describes the experiments ran which are then discussed in Section V. Finally Section VI draws the final conclusion giving a glimpse of possible future work.

II. Background

The problem of reward scarcity is central in game AI, it is well known the case of the game Montezuma's Revenge, part of the ALE. This game offers very sparse rewards to the player and only when performing specific actions. Pohlen et al. [8] managed to improve the state-of-the-art performance of general RL algorithms by designing an algorithm based on heavy exploration. For Ecoffet et al. [9] reached superhuman performance but with human-provided domain knowledge into the learning process. Others have reached superhuman performance but only by using demonstration-based approaches by either providing YouTube videos of human players [10] or a single successful demonstration [11].

A different approach to the aforementioned ones is to enhance the reward signal providing additional rewards using domain knowledge: *reward shaping*. This approach tries to identify functions that simplify the learning problem while yielding the same optimal policy as result. Given the typical definition of Markov Decision Process (MDP) as a tuple of: S, set of states; A, set of actions; p, transition function between states; r, reward function. In [12], the authors define a potential-based shaping function as $F(s,a,s') = \gamma\Phi(s') - \Phi(s)$ with $\Phi : S \to \mathbb{R}$ and $\gamma \in [0,1)$. Such function is used to define a new reward function $r' = r + F$, i.e. sum of the original reward function and a potential-based shaping function. The above definition is proven to be necessary and sufficient for guaranteeing that the optimal policy learned is equivalent to that of the original MDP using r. However, Hu et. al. [13] show, starting from an unconstrained shaping function f, how it is possible to adjust its signal to learn an optimal policy. The new definition $F = z_\phi(s,a)f(s,a)$, introduces the shaping weight function z_ϕ to properly weight the contributions of f. When using multiple shaping functions, z_ϕ can return a vector of weights to combine them together.

The idea of capturing the dynamics of a game has been explored by taking different approaches, some borrow concepts from psychology, some rephrase the reinforcement learning problem for transfer learning while others model the incentive for exploration as a measure of new dynamics discovered. The following paragraphs provide insight on each one.

Holmgard et al. [14], use the concept of *affordance* to describe specific player-styles or behaviours. The concept of affordance is closely related to the question "what can I do with this object". This positions the concept close to fields that make interaction their focus (e.g. Human Computer Interaction and branches of Robotics). In [14] the authors associate a metric to each affordance, then design AI agents that plan their actions to maximise/minimise a selection of such metrics in order to show specific behavioural traits. These agents were created in order to develop a portfolio of *personas* to be used for playtesting purposes.

Perez et al. [15] have developed an MCTS modification promoting the exploitation of actions that bring the player in game states that trigger new and unseen interactions. This is done by collecting statistics on new interactions during the *rollout* phase of MCTS. Their experiment showed that using information coming from the dynamics of the environment can bring significant performance improvement.

In the field of Reinforcement Learning the concept of *successor feature* has been described in [16]. Based on the work of Dayan [17] and the above definition of MDP, the reward function from state s to state s' performing action a can be redefined as $r(s,a,s') = \boldsymbol{\phi}(s,a,s')^\top \mathbf{w}$, where $\boldsymbol{\phi} \in \mathbb{R}$ and $\mathbf{w}$ is a vector of weights. The function $\boldsymbol{\phi}$ yields a vector encoding the dynamics of the state transition. As a consequence a new description of the action-value function q is derived applying the new form of $r(s,a,s')$ to its definition: $q(s,a) = \boldsymbol{\psi}(s,a,s')^\top \mathbf{w}$. The function $\psi(s,a,s')$ will decouple the dynamics of the environment from the reward function in the shape of *successor features*. In this paper we use events to produce such a set of features.

Finally, we want to highlight that player behaviour arises from a combination of several factors, the following are the two most relevant: (1) the game specification, i.e. rules and content of the game (e.g. cards, boards, dice etc.); (2) the ability of the player to search the game tree. When looking at player behaviour from the perspective of (1) we can analyse the *behaviour possibility space*, how the game limits the actions of the player in the game. Whereas from the point of view of (2) we can look into the *behaviour expressivity space*, how the reasoning abilities of the player limit their own behaviour. Opponents/teammates can also heavily affect it but it can be regarded as a secondary factor as not all games are multiplayer. For example, [18] addresses the behaviour possibility space since it manipulates the decks (content of the game) but with a fixed AI player. However this research works in the direction of agents with high behaviour expressivity which is a trait we deem necessary for solid automated playtesting.

A. Rinascimento

Rinascimento (also stylised $\mathcal{R}$) is a framework[1] for the development of Game AI, it is based on the popular board

[1] github.com/ivanbravi/RinascimentoFramework

game Splendor[2] published by Space Cowboys and designed by Marc André [19]. Splendor is a turn-based multiplayer board game, the objective is reaching 15 prestige points first, points are obtained by either buying bonus cards purchased using tokens or attracting nobles based on the cards purchased. Four cards are laid face-up from each of the three decks, cards can be bought or reserved and bought later, in both cases they move to the player's hand. When a card is bought the player receives the card's prestige points, then a new card is drawn from the same deck and placed face-up on the table. A card can be reserved if it is face-up or the first face-down of its deck, in this case it will be revealed to the other players only when purchased. A player can pick either three tokens of different suits or two of the same but only if there are more than 3 in its stack. Each player must perform a single action each turn and never own more than 10 tokens.

The framework can be used to play Splendor-like games, in fact, the parameters in the game's rules are exposed so that they can be easily changed (see [19]) for a complete list. The same applies to the decks of cards in the game, Splendor has 3 decks but $\mathcal{R}$ can support any number of decks.

$\mathcal{R}$ implements a Forward Model (FM) that can be used by planning algorithms to simulate future game states given the actions performed by the players. The use of a FM doesn't affect the real game state but it provides "oracle" skills to the agents during their decision making. It has been used to test SFP algorithms [19] and particular attention was given to hyperparameter tuning. The experiments have shown that algorithms using prestige points increment as reward signal, prefer very short action sequences when planning. This is likely due to the highly stochastic nature of the game coming from decks shuffling, partially observable game states and opponents actions.

Even though $\mathcal{R}$ can potentially express a multitude of Splendor-like games in this paper we will mainly deal with the two-player version of the original game, unless otherwise stated. The nature of the game makes enumerating the legal action-set computationally expensive (see [19] for a more detailed explanation linked to the parametric nature of the framework). To avoid this enumeration, $\mathcal{R}$ provides the tools to randomly sample the action space with the option of controlling the sampling through a unique random seed. Such randomly-generated actions can be illegal if performed in a different game state, that's why extra care needs to be taken when designing the agents.

B. Game-playing Agents

Statistical Forward Planning algorithms can be used in games where a Forward Model is available, this will allow the algorithm to simulate future actions without affecting the current state of the game. Both Rolling Horizon Evolutionary Algorithm and Monte Carlo Tree Search have shown remarkable performance in games-based competitions such as the GVGAI Planning Competition and the Fighting Game AI Competition [20]. In [19] the authors have implemented a number of game-playing algorithms:

[2]boardgamegeek.com/boardgame/148228/splendor

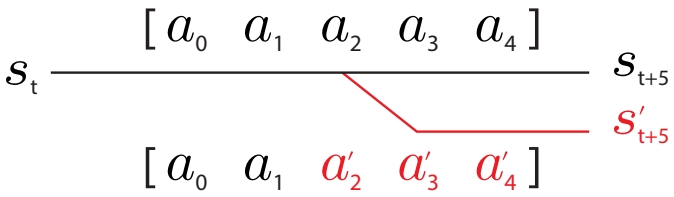

Fig. 1. Branching mutation on a sequence with 5 actions. Actions are copied up to the mutation point a_2, the following actions are then picked randomly.

- RND: random actions;
- OSLA: one step look ahead agent;
- MCTS: an implementation of MCTS using the Upper Confidence Bound (UCB) formula for the node *selection* and Iterative Widening for dealing with the unknown size of the action space in the *expansion* phase;
- BMRH: Branching Mutation RHEA evolves a population of action sequences, during the mutation phase a point in the sequence is selected and from there on the actions are mutated with new legal random actions by rolling the state forward.
- SRH: Seeded RHEA evolves an action sequence made of seeds thus circumventing the issue of actions becoming illegal. The seeds are fed to the action random sampling thus fixing the actions generated.

The hyperparameters of the three different SFP agents have been tuned for optimal performance against the OSLA agent. In this paper tuned agents will be characterised with an asterisk (*) after their name (e.g. BMRH* for the BMRH hyperparameter space). In this work we will use the BMRH agent and its hyperparameter space described in [19] for two main reasons: it has proven similar peak performance as MCTS and its hyperparameter space resulted much denser of well performing configurations than MCTS and SRH.

1) Branching Mutation Rolling Horizon: BMRH is a type Rolling Horizon Evolutionary Algorithm introduced in [19], for every game tick it evolves sequences of actions evaluating them using a forward model. Such evaluation is based on some value function that measures the quality of the sequence, it is usually the score increment from the starting to final state. Its main feature is the mutation operator: during the creation of a new offspring the original sequence is copied one action at a time and also played until the mutation point, from there on new actions are sampled and added to the sequence until the end of the sequence. A visual representation of the process is shown in Figure 1. At the end of the evolution it picks the best sequence and returns the first action.

The actions in a sequence are dependent to each other in a cascade fashion: an earlier action could make a later action illegal. Given the tight constraints of this game on legal actions (all resources are very different and limited) we implemented the branching mutation in order to reduce the impact of illegal actions to the evolution. For the same reason no crossover operator is used in creating new offsprings. See [19] for further details and a list of it hyperparameters.

C. Hyperparameter Tuning

Most algorithms have several parameters that can be adjusted offline to modify its online execution. Examples for RHEA are the mutation probability or the action sequence length. Such parameters are called hyperparameters and in scenarios like game-playing they can have a big impact on the agent's performance.

There's a vast number of applications of hyperparameter tuning in many academic fields, however in the field of game AI its application is still limited. In [21], Lucas et al. have compared several optimisation algorithms in the task of tuning a game-playing RHEA AI and concluded that the N-Tuple Bandit Evolutionary Algorithm (NTBEA) is the best. NTBEA, introduced in [22], is a model-based optimisation algorithm, it builds a model of the hyperparameter space using the information gathered by the fitness evaluations of the hyperparameter candidate solutions. This information is stored in multi-armed bandits where each arm of a bandit is a configuration of N specific hyperparameters. Such bandits use 1-, 2- or N-tuple models depending on the algorithm configuration, this means that the outcome of the simulations will be stored and aggregated according to single, couples or triplets of hyperparameters respectively.

III. Value Functions

In Game AI (or more broadly in RL), a value function is a model that assigns to a given set of features a value expressing its quality. In the following we will present two sets of features: state features and events features. These represent two different but related aspects of gameplay its static and dynamic representations respectively.

A. State-value Functions

The classical approach to create value functions for playing games is using a function approximator (e.g. an Artificial Neural Network (ANN)) to evaluate the expected mean reward associated to a state. This function is typically called $V(s)$ where s is the game state evaluated. In order to have a general approach in the $\mathcal{R}$ framework, we present in the following the parametric grammar used to encode the game state regardless of the game specification. In addition to being parametric, this grammar is also size-invariant, meaning that given a set of $\mathcal{R}$ game parameters, the size of the feature vector encoding the state will always be of the same size. The fixed size allows methods like feed-forward ANN or other regression models to work properly.

The grammar is comprised of the rules in Table I *CardsRemaining* is the counter of cards left in the deck. *TokensAmount* is a vector of the amount for each token type (order is consistent). *Suit* is a one-hot encoding with as many bits as suits in the game. *Cost*, similarly to *TokensAmount*, is a vector of the cost for each token type. *CardSuitCount* is the amount of cards, divided by suit, that a player has bought.

Between squared brackets we specified the items whose number changes depending on the game's parameters. Given this grammar, the game state of the 2-player version of Splendor has 144 dimensions.

TABLE I
List of all the production rules for compiling the game state. Terminal symbols in italics.

Symbol		Production rule
State	->	Board [Player]
Board	->	[Deck][Noble] *TokensAmount*
Deck	->	*CardsRemaining* [Card]
Card	->	*Suit Cost Points*
Noble	->	*Points Cost*
Player	->	*Points TokensAmount CardSuitCount* [Card]

B. Event-value Functions

In a RL scenario, where S is the set of states and A is the set of actions, our main objective would be learning a value function. This can be a state-value function in the shape of $v : S \to \mathbb{R}$ or an action-value function as $q : S \times A \to \mathbb{R}$. However in this work we introduce the concept of event-value function: $h(E_{s \to s'})$ where s is the current state, s' is a future reached playing $\mathbf{a}$ (sequence of n actions), $E_{s \to s'}$ is the set of events happening between s and s' as E is the set of all possible events. This function requires a model $m : S \times A \to E$ that generates the events triggered by a from s.

As a first step in developing this approach we have deconstructed the function h as $h_{\mathbf{w}}(E_{s \to s'}) = f_{\mathbf{w}}(\sigma(E_{s \to s'}))$. The function σ synthesises $\boldsymbol{\theta} \in \mathbb{R}^t$ features from the list of events $E_{s \to s'}$ while $f_{\mathbf{w}}(\boldsymbol{\theta})$ is a parameterised *function model* in $\mathbf{w} \in \mathbb{R}^t$ weights.

This separation between features $\boldsymbol{\theta}$ and weights $\mathbf{w}$ perfectly isolates respectively the identification of meaningful gameplay features from the multi-objective optimisation problem of prioritising certain events dynamics.

1) Event Logging: In $\mathcal{R}$ the game state is made up by the following elements: decks of card, face-up cards, nobles, common tokens, joker tokens, player hands. Each player state is made up by: points, purchased cards, common tokens, joker tokens, reserved cards, hidden reserved cards. Whenever the engine performs an action that modifies the game state, this generates an event which is forwarded to a list of subscribed loggers. An event is described by the fields: *tick*, when it happened; *who*: who triggered it; *type*, unique type identifier in the range $[0, \#types - 1]$; *duration*, how long it lasted; *durationType*, whether the event is instant, delayed or durative; *attributes*, dictionary of attributes characterising the event; *signature*, list of possible attribute keys; *trigger*, what action triggered it. This description provides very rich information that can be used by the player to make better informed decisions. This definition is general enough to be applied to most games, in fact, it was compiled by referring to several AI game-playing competitions specifically the knowledge they provide to the AI players. However, in this work we explore the use of just two fields: *who* and *type*. For the specific case of $\mathcal{R}$ we have defined 18 different event types, see Table II. The column $\mathbf{Type}^{id}$ assigns a unique id to each event. $\mathbf{Type}^{hc}$, instead, groups the events in 5 hand-crafted macro-events and filters out minor events (id = -1). In particular, when it comes to token related events, an event is triggered for each single token suit.

TABLE II
LIST OF ALL THE EVENTS, P_i IS THE i-TH PLAYER, E FOR EVENTS TRIGGERED BY THE PASSIVE RULE OF $\mathcal{R}$'S ENGINE. WHEN A STATE ELEMENT HAS SEVERAL EVENTS THESE ARE LISTED IN THE EVENT COLUMN SEPARATED BY A COMMA AND SO ARE THE RELATIVE IDS.

State element	Event	Who	$\mathbf{Type}^{id}$	$\mathbf{Type}^{hc}$
Noble	place, take, receive	P_i	7, 0, 14	-1, -1, 3
Table's token	increase, decrease	P_i	1, 2	-1, -1
Table's joker	increase, decrease	P_i	3, 4	-1, -1
Table's card	draw, place	P_i	5, 6	-1, -1
Player's token	increase, decrease	P_i	8, 9	0, -1
Player's joker	increase, decrease	P_i	10, 11	0, -1
Table's card	reserve, hidden	P_i	13, 12	2, 1
Player's points	from card	P_i	16	4
Player's points	from noble	E	17	4
Player's card	buy	P_i	15	-1

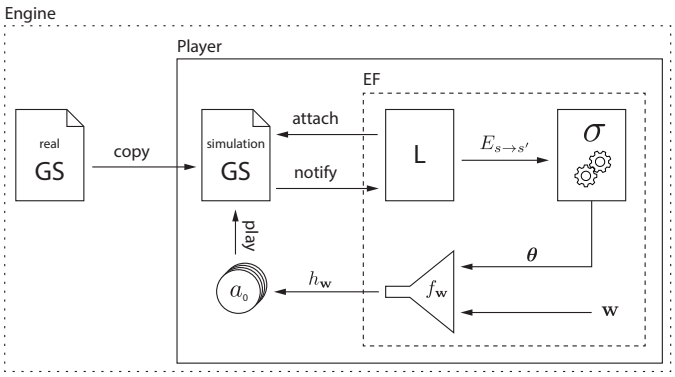

Fig. 2. Interaction between engine, state, player and EF. The player starts simulating the action sequence from action a_0. At the end of the evaluation the EF can be queried to get the value v.

2) Logging dynamics: An EF contains two components: a synthesis function σ, an event logger (L) and a set of weights w. The Event Logger receives all the events triggered by the game state (GS) it is attached to. Then, during the action sequence evaluation, the events generated are forwarded to the synthesizer. σ is responsible for filtering and processing the events in order to produce a vector of features $\boldsymbol{\theta}$ of the same length of $\mathbf{w}$. As the EF-based player prepares to evaluate an action sequence, the EF subscribes to the events generated by the game state used for the forward planning, then the actions are performed. At this point, the EF can compute the value $v = h_{\mathbf{w}}(\boldsymbol{\theta})$. See Figure 2 for a schematic representation of these dynamics.

3) Synthesis Function: Our synthesis function σ is quite straightforward: it counts the events grouping them by *type* filtering out the events not triggered by the player. Currently, events are filtered directly in the code by comparing the player id, however this could be learned as well introducing opponent critique. In order to reduce the size of the feature vector we also introduce the possibility of remapping the types to group them. See Table II for the mappings, id is the identity mapping while hc is a hand-crafted mapping. Note that mapping to -1 is a way of discarding the event altogether.

4) The implementation burden: All these functionalities come at a cost, creating and embedding into a game a logging infrastructure together with model and synthesising functions requires time and engineering skills. However we need to make two observations to put things into perspective.

First, this approach has been developed with the long term objective for being applied in a playtesting scenario where there is an explicit need for expressing as many strategies as possible in the most controlled way. Take the example of an agent based on an ANN that given the current state as input provides next action to play. The control we have over such agent, and consequently its strategy, is by adjusting the ANN's weights, unfortunately selecting the weights is a very delicate task. In a EF instead there's a direct link between game dynamics (represented as features) and the agent's behaviour.

Second, what is currently done by hand could be done by a specialised model trained using classic RL techniques or engineered in a general-purpose game engine. In the former, the RL model would receive as input the starting and ending states and output the features vector. There's a trend that is moving from single network architectures to multi-network architectures, i.e. from the Deep Neural Networks in [3] to AlphaStar [23]. This forces each portion of the system to focus and specialise into a specific task. What we are envisioning is the possibility in the near future to automatically generate the features $\boldsymbol{\theta}$ by learning the mapping we are currently hand-crafting in $\sigma(E_{s\to s'})$ with the aid of the logging system. In the latter the implementation overhead would be extremely limited, potentially limited as a one-liner call to an embedded logging engine.

C. Function Models

In an effort to keep the complexity of this study under control and preserving interpretability we use three models: *linear*, *multivariate polynomial*, *Pruned Artificial Neural Network* (PANN). Regardless of the model the agent simulating an action sequence from state s to s' computes its fitness signal as $f_{\mathbf{w}}(s') - f_{\mathbf{w}}(s)$.

1) Linear: function simply consists in summing each feature multiplied by a weight, i.e. $lin_{\mathbf{w}}(\boldsymbol{\theta}) = \mathbf{w} \bullet \boldsymbol{\theta}$

2) Multivariate polynomial: function of degree d, see Equation 1 where the function sel, selects the j-th element in the i-th d-multicombination of $|\boldsymbol{\theta}|$ variables

$$poly^{d}_{\mathbf{w}}(\boldsymbol{\theta}) = \sum_{i=1}^{\binom{|\boldsymbol{\theta}|}{p}} w_i \prod_{j=1}^{d} sel_{\boldsymbol{\theta}}(i,j) \tag{1}$$

For example given $\boldsymbol{\theta} = [\theta_0, \theta_1]$ we have $poly^2_{\mathbf{w}}(\boldsymbol{\theta}) = w_1\theta_0^2 + w_2\theta_0\theta_1 + w_3\theta_1^2$. We want to emphasise that we don't need any constant w_0 since $h_{\mathbf{w}}$ is a ranking function. The purpose of using a higher degree function is to detect possible dependencies between features.

3) PANN: function approximation model based on an Artificial Neural Network with pruned connections [24]. Pruning consists in nullifying a percentage of the connection weights in a fully connected feed-forward topology, it has shown comparable performance to using the full network but saving a considerable amount of computation. See Figure 3 for an example. These name these models as $pann^{p\%}_{\mathbf{w}}$ where $p\%$ is the percentage of active connections. The decision of enabling or nullifying a connection is controlled by a random seed.

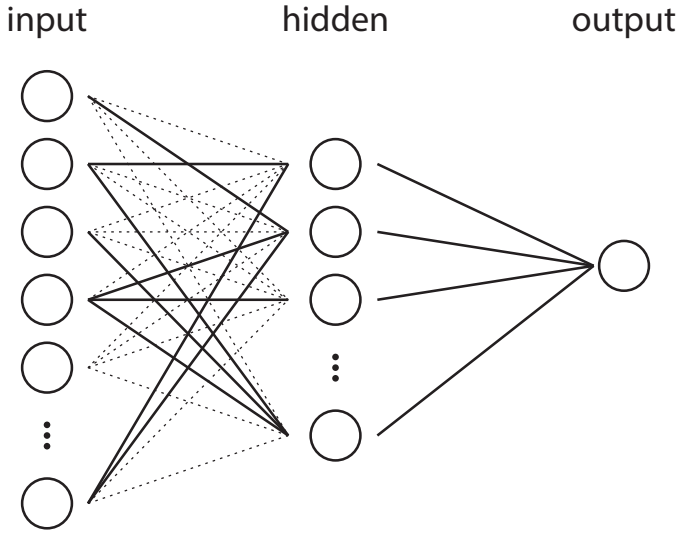

Fig. 3. topology of the pruned feed-forward ANN, the dashed lines represent the pruned connections while the solid lines the active ones. The connections to the output node are all preserved active.

When using a PANN the values of the feature vector fed as input is rescaled to guarantee inputs within $[0, 1]$ simply by dividing all the features by fixed number for consistency across games.

The space of possible weights $\mathbf{w}$ then becomes the hyperparameter space for the function model whether that's used by a SF or an EF. In machine learning this would be considered a simple parameter space, however, in SFP, these parameters don't belong to the model built by the game-playing algorithm making them hyperparameters. This is in contrast with the classical applications of state-value functions which are learned. Unfortunately learning an SF is usually an expensive task and to provide a level ground and experimental parity we will treat both SF and EF models as hyperparameter spaces.

Compared to [7] we introduce the use of PANN to provide State-value Functions with an appropriate model to support a fair comparison between the two approaches. PANN are particularly appropriate to maintain the complexity of the function model relatively low, i.e. the amount of weights to tune. This is particularly important remembering the amount of features used by the SFs.

TABLE III
TABLE SHOWS THE NUMBER OF WEIGHTS FOR EACH FUNCTION MODEL FOR SF AND EF. THE FEATURES COLUMN SPECIFIES HOW MANY EVENT FUTURES ARE PROVIDED, WHILE THE TOTAL COLUMNS INDICATES HOW MANY WEIGHTS THE FULLY CONNECTED NETWORK WOULD HAVE.

Event-value Functions

Model	**Features**	$\lvert\mathbf{w}\rvert$
$lin^{hc}_{\mathbf{w}}$	5	5
$poly^{2,hc}_{\mathbf{w}}$	5	15
$lin^{id}_{\mathbf{w}}$	18	18
$poly^{2,id}_{\mathbf{w}}$	18	171

State-value Functions

Model	**Total**	$\lvert\mathbf{w}\rvert$
$lin_{\mathbf{w}}$	202	202
$pann^{10\%}_{\mathbf{w}}$	5760	576
$pann^{5\%}_{\mathbf{w}}$	5760	288
$pann^{1\%}_{\mathbf{w}}$	5760	58

IV. EXPERIMENTS

The main objective of the experiments is to show that event-value functions can be a valid substitute to both state-value functions and score-based state evaluations for SFP algorithms. We designed three sets of experiments:

- Hyperparameter Tuning: hyperparameter tuning using NTBEA (see Section II-C) in the hyperparameter space of BMRH coupled with the hyperparameter space of an EF or a SF, run for several EFs (see Section IV-A). The purpose is to understand the theoretical potential of each hyperparamenter space;
- Round-robin Tournament: picking the best agents tuned with NTBEA we are going to set one against the other (see Section IV-B);
- Multi-opponent Games: compare the stability in terms of win rate in scenarios where there are 3 opponents (see Section IV-C).

All the SFP agents in the following experiments were allowed a budget of 1000 simulated actions for each turn of the agent. In the following sections we will address the tuned agents with an asterisk (*) and the hyperparameter space without it.

A. Hyperparameter Tuning

Both event- and state-value functions are parameterised in a set of weights $\mathbf{w}$ (whether these are used in a linear, polynomial or ANN model), these define an hyperparameter space of $|\mathbf{w}|$ dimensions. Since NTBEA is a discrete optimisation algorithm, we discretise the continuous space of each weight w_i as $\mathbf{W} = \{-1, -0.8, -0.6, -0.4, -0.2, 0, 0.2, 0.4, 0.6, 0.8, 1\}$. The limited variety of weights could constrain the ability of fine tuning the heuristics, but it will most likely be sufficient for the lower dimensional $\mathbf{w}$.

When the BMRH agent uses an EF (or a SF), BMHR's and the $\mathbf{w}$ hyperparameter spaces are combined to define the hyperparameter space of BMRH+EF (or BMRH+SF), combining the two spaces will allow the two components to adapt to each other. We are going to tune several configurations of BMRH+EF and BMRH+SF to perform as well as possible against BMRH*, therefore NTBEA will be set to maximise the win rate of the tuned agent in a 2-player Splendor game. For each set of hyperparameters evaluated, we run a single game in spite of the high stochasticity of $\mathcal{R}$, in fact, NTBEA was designed to deal with objective functions with high noise.

In Table III are reported all the models used in the experiments, when considering the dimensionality of the hyperparameter space for those experiments we need to add 10 dimensions from BMRH to the dimensions specified in the table. NTBEA was run 100 times for each tuning configuration and with several budgets: 50, 100, 200, 500, 1k, 10k, 100k, 500k games. Once the budget is over, the suggested optimal configuration is validated on 1000 games against BMRH*. Under the same conditions we also tuned the BMRH, but with the classic score-based value function, as this will give us an idea of the trade off between tuning a basic agent with a small hyperparameter space (10D) and a more sophisticated agent with a much larger space (from 15D to 576D). NTBEA has two hyperparameters that can be adjusted, we picked the

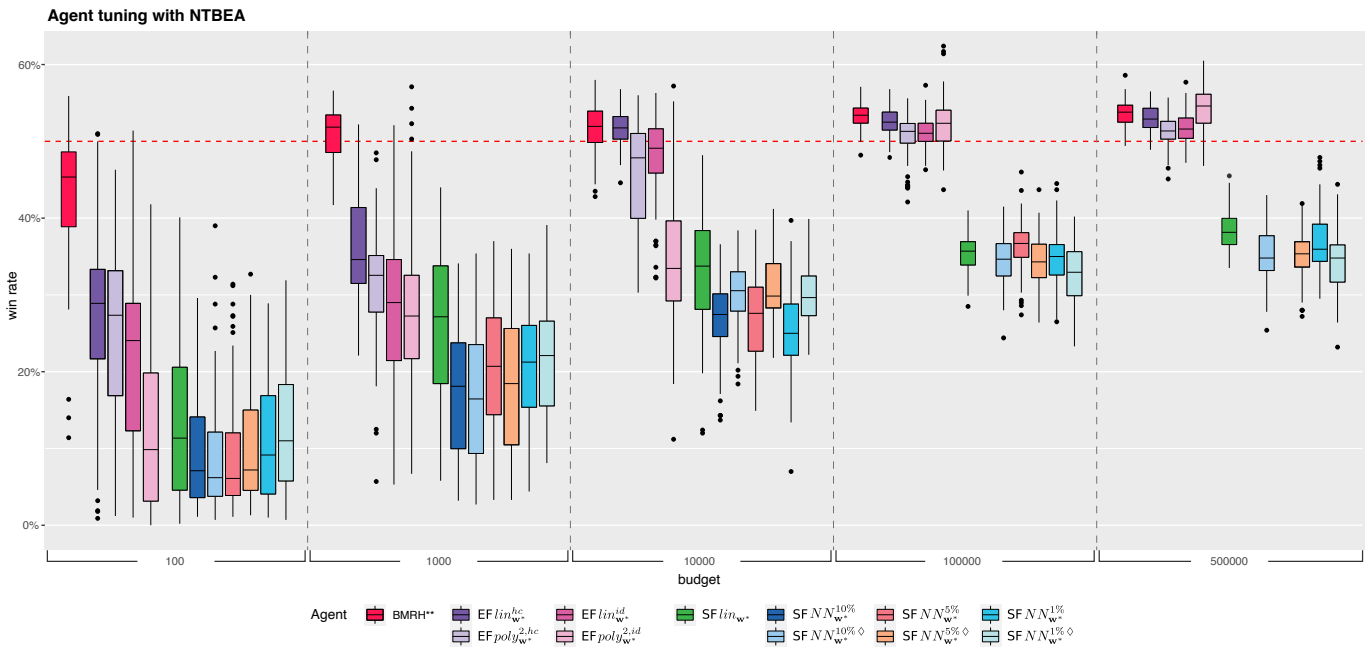

Fig. 4. NTBEA results all the configurations of BMRH+EF as the budget variables. When data is missing, it is due to exeeding the maximum computational time allowed.

values $k = 1$ and $\epsilon = 0.7$, see [22] for more details. The results from the validation phase after NTBEA are shown in Figure 4.

A few further notes must be made for the PANN model. We have set the seed to be the module 10 of the experiment id (1-100), this was a trade-off between trying several network configurations and still run NTBEA with the same exact configuration several times. The combinatorial nature of the tuning algorithm makes tuning large spaces challenging as the number of simulations grows as well. We were limited by maximum amount of computational time of 10 days to run each instance of NTBEA. For this reason we PANN models were tuned using NTBEA with all three tuple models and just 1- and N-tuple models, this last scenario can be discerned by a $\Diamond$ in the (e.g. $NN^{10\%}_{\mathbf{w}}$ uses 1-, 2- and N-tuples whereas $NN^{10\%\,\Diamond}_{\mathbf{w}}$ only 1- and N-tuples)

B. Round-robin Tournament

The main purpose of running a round-robin tournament is to get a better understanding of the all-round performance of the optimised agents, not only against the reference agent BMRH*. From the previous NTBEA experiments we selected the best (highest win rate) configuration according to the validation statistics, we are going to name this configurations by using the subscript $\mathbf{w}^*$ instead of the generic $\mathbf{w}$ (e.g. $poly^{3,hc}_{\mathbf{w}^*}$). Instead BMRH** is the optimal BMRH tuned against BMRH*. In Figure 5 the win percentage of each couple of agents is reported based on 10000 games of two-player Splendor. This number of samples guarantees that the real value of the estimated win percentage will lie within a 95% CI with boundaries ± 1.

C. Multi-opponent Games

Since one of the higher sources of stochasticity in $\mathcal{R}$ are the opponents, we tried to evaluate the robustness of the agents by varying the number of opponents. We first let each tuned agent play against 3 RND agents in a 4-player Splendor game for 1000 times. This is to evaluate if the presence of a random player can influence the their performance. Then we repeat the experiments but with 3 BMRH* opponents. Figure 6 shows the outcome of these experiments.

V. Discussion

A. BMRH+EF Tuning

The tuning experiments have highlighted a multifaceted scenario concerning both the optimisation algorithm and the BMRH+EF/SF spaces. The results reported in Figure 4 show the box plot summarising the 100 runs for each experimental condition, outliers have been reported with single dots. As a sanity check we can notice the average fitness improving as the budget increases and the box-blot whiskers shrinking highlighting a more solid convergence of NTBEA.

1) Overall considerations: Our baseline, the BMRH hyper-parameter space can be tuned to outperform BMRH* using a budget of 1000 games, this is a very small amount of computational resources especially for such a fast framework. As the budget increases the performance saturates never crossing the 60% mark. The best BMRH configuration is found with a small budget of 500. This suggests that NTBEA could either make a better use of the budget pushing for more exploration, or have an intrinsic limit in dealing with big amounts of data in small search spaces.

With regards to the BMRH+EF spaces, in order to find well performing solutions, NTBEA needed a budget of 10k

games for $lin^{hc}_{\mathbf{w}}$, this was expected as the search space has 5 more dimensions. It is actually reassuring as it took only 10 times the budget use by BMRH for a search space $> 10^5$ times bigger. The best overall performance, with a win rate of 62%, was found using the $poly^{2,id}_{\mathbf{w}}$ EF and a budget of 100k. This is an outlier considering the average outcome for such experimental condition. However the purpose of these experiments is not evaluating NTBEA but rather the possibility for EFs to improve BMRH performance, thus outliers are just as relevant as any other data point.

Finally the BMRH+SF spaces show a healthy increase in performance with growing budget but they don't show promising nor competitive performance. Only a few outliers get close to the 50% mark (SF $NN^{10\%\,\diamondsuit}_{\mathbf{w}}$). Even though the size of those spaces is comparable to that of EFs, it's likely the dishomogeneous nature of the features (scalars vs one-hot) that severely complicates the tuning process. Two other limiting factors could be the number of connections used and the size neurons in the hidden layer, however this had to be restrained to comply to the limited computation available.

With both EFs and SFs we can notice how the performance reached by models with different amounts of weights shrinks as the budget grows, highlighting how on average even simple models can yield comparable results, only

We want to emphasise that with the current approach on EFs, even though we can recognise score variations with events 16 and 17, we are not actually using their magnitude. So they are not able to differentiate between a variation of 3 points and that of 1. The same is true for the amounts of coins taken or given. This is an area of improvement that will require to query information stored in other fields of the event data structure. In this work, however, we stuck to the simple use of *type* and *who* fields of an event, for the sake of simplicity. On the other hand, SF can potentially access all this extremely valuable information as it is explicitly embedded in the state representation. However the tuning process is likely hindered by the sheer complexity of the state representation burying useful information in the midst of mostly-irrelevant information.

2) Optimal configurations: One of the most interesting aspects of these new EFs is how they can be used to better understand the game and the player's strategy. Both $lin_{\mathbf{w}^*}$ can be used to directly infer the relevance (or the agent's preference) of the events from *hc* and *id*. Unfortunately SFs are more complicated to address as their multi-layer nature compounds features in a non-trivial way.

Inspecting $lin^{hc}_{\mathbf{w}^*}$ optimal weights (0.2, 0.2, -0.4, -0.6, 0.8) and matching these with Table II we can notice that taking tokens is positive (0.2) and even more important if different token types are taken as the contribution scales with the number of events triggered; player's points are the most important (highest weight: 0.8); reserving cards is mildly discouraged (-0.4) unless if hidden which is lightly encouraged (0.2); attracting nobles is seen as a negative event $w_3 = -0.6$, however when considering that this event is always triggered together with the event linked to $w_4 = 0.8$ it becomes apparent that this is just a way of preferring point events that come from cards. This last consideration in particular shows how this new

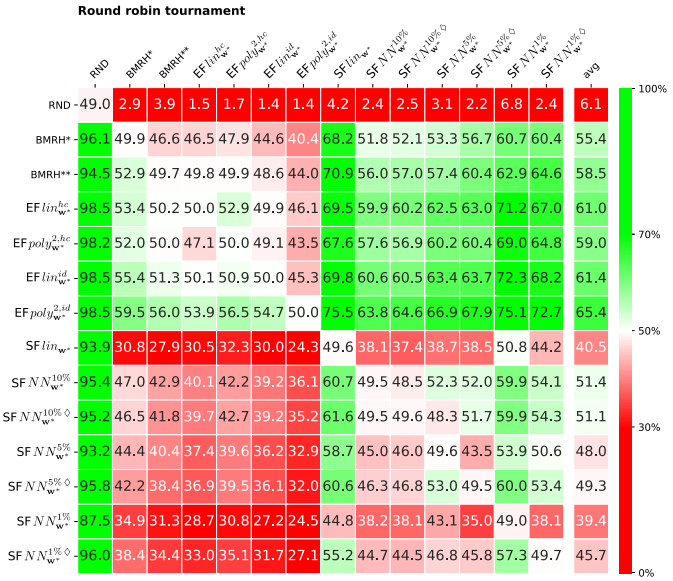

Fig. 5. Results for the round-robin tournament, each element $(row, column)$ shows the win rate of player row against player $column$. The heatmap colours are invariant in 0-30% and 70-100% in order to highlight relevant subtleties around the 50% mark. The last column, *avg*, shows the average win rate across all opponents.

EF approach is able to differentiate between action sequences leading to the same score variation.

These results are the demonstration that using an EF the agent can express very specific and refined strategies in terms of in-game behaviour. Unfortunately interpreting the results for the $poly_{\mathbf{w}^*}$ EFs can be very complicated, the optimal configurations can be found in the online repository[3].

The algorithm hyperparameters can give further insight on the agents. We saw from [19] that all the agents were tuned with a sequence length of 2 actions. This time, instead, several of the optimal configurations had a sequence length of three actions: BMRH**, $poly^{2,hc}_{\mathbf{w}^*}$ and $poly^{3,hc}_{\mathbf{w}^*}$. These longer horizons highlight the need for longer planning in order to beat BMRH*. On the contrary, all SFs have an horizon of a single action. This highlights the inability of tuning a value function able to analyse the compound contribution of several actions. This may be due to the inadequacy of the models or the tuning getting stuck in a strong local minimum.

B. Round-robin Tournament

The heatmap in Figure 5 immediately suggests a staggering performance difference between EFs (top half) and SFs (bottom half), we can see BMHR*/** and EFs beating SFs in the top-right corner. This is also suggested by a higher average performance across opponents (reported on the *avg* column). Our baseline is BMRH** shows an improved tuning compared to BMRH*, as they share the same hyperparameter space. However when playing against an EF-BMRH agent, it shows a 50/50 win rate at best. The result that immediately stands out is the performance of $poly^{2,id}_{\mathbf{w}^*}$, this agent dominates all the other agents, in particular BMRH** with a solid 56.4%

[3]at the project path *RinascimentoFramework/agents/journal.json*

of victories. This result is encouraging as it shows a promising edge over a points-based agent.

Between EFs we see a fairly clear dominance of $poly^{2,id}_{\mathbf{w}^*}$ over all the agents in the tournament suggesting that using a more fine-grained event set can better characterise the in-game behaviour of the agent. Instead, SFs suggest that enabling 10% of the connections yields the best results between SFs. Also in this case we can see how when tuning we must carefully weight the amount of weights against its computational burden, vastness of the search space and quality/amount of the features available to the function.

What is striking, looking at these results and at NTBEA's, is the lack of a really strong player with win rates around 80/90%. There is a number of reasons for this. The most likely reason is the heavy interference of stochasticity in the game which can suddenly turn the tables. Another reason is the limited amount of budget allowed to the agents (i.e. 1000 simulated actions per turn). This is a limiting factor in their ability to predict the future, however, the amount of stochasticity could also just be prohibitive. Finally, we must remind that the real search space for the weights is continuous, thus the limited set of weights used to discretise such space might be limiting the NTBEA's ability of fine tuning the functions with higher number of weights.

C. Multi-opponent Games

In these two experimental settings we evaluated the win-rate drop (Δ) between playing against one or three opponents. In the first case, playing against RND agents, we can notice how SFs mixed results with most agents in the range (10-23%), the exceptions are SF $lin_{\mathbf{w}*}$ with the worst overall performance and SF $NN^{1\%}_{\mathbf{w}*}$ the only SF agent getting close to EFs consistency who all remarkably have a very slight drop between 0.3% and 2.1%. BMRH*/** have a relevant drop but still contained compared to most SFs. When playing against multiple BMRH*, we can see an even clearer separation between EFs and SFs with the former showing fairly solid performance (in 9.9-18.5%) with a worst win-rate of 36.3% while the latter collapsing (Δ in 22.7-38.5%) with a best case scenario of 11.1% win-rate. These results show how on average the EF-based agents can guarantee a much more consistent performance in more noisy and competitive games than if they were using point heuristic or a state-value function model.

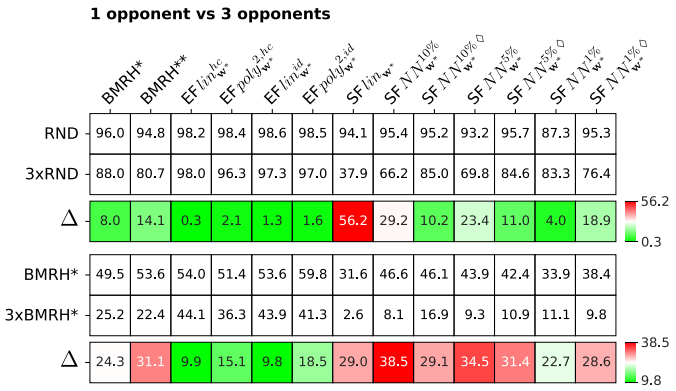

Fig. 6. Win rates for the column players against the row agents. In each one of the two matrices, the third row represents the delta between the win%. For a 4-player game the uniform random target for the win rate is 25%. Since we are comparing 2 and 4-player games we are interested only in the win rate and not in relative placement (2nd, 3rd, 4th).

VI. Conclusions

In this paper we have presented a further analysis of a novel approach for value functions in scenarios with scarce or absent reward signals: event-value function $h_{\mathbf{w}}(E_{s\to s'})$. The playing field is that of easy-to-tune Statistical Forward Planning algorithms. The main purpose is to provide a smoother gradient in the evaluation of a sequence of actions. This evaluation will be based on the set of events that are triggered during the execution of such actions. These events are characterised by a type. Discerning by type, we can count the number of events that happened and create a feature vector. This feature vector describes the dynamics of the system uncovering what happens in the state transitions. Finally by recombining and weighting the features we can evaluate the quality of the dynamics triggered by the action sequence.

The EFs approach has shown performance improvements in terms of win rate when compared to the baseline agents BMRH* and BMRH** whereas SFs were fare from achieving comparable results even to our baselines. Event-value functions have also proven more robust when playing against more opponents, as shown in Section V-C. EFs also allowed a more controllable characterisation of the game-playing style of the agent as shown by the EF's weights analysis. Using an event-value function we can discern the difference between two action sequences that lead to the same outcome in terms of score variation, meaning we have finer control over the behaviour of the agent. This particular feature is very important when it comes to the ability of automatically play-testing a game, in fact optimal agents are blind to the variety of possible non-optimal but human-like strategies in the game.

This novel methodology was tested in a relatively simple multiplayer board game with limited use of domain knowledge. We have seen how simplifying the event mapping, from *id* to *hc*, can bring a remarkable speed up in tuning time. Such flexibility provides a great oppurtunity for designers and developers to inject domain knowledge in the form custom-designed events or event filtering. With further development of the methodology, it could be fast and effective even in complex real-time games with appropriate involvement of the designers.

VII. Future work

For future work we plan on doing a variety of improvements and enhancements to the work presented here and also to apply this to different scenarios.

Improvement could be brought by the use of the information from the event, this far, only the *type* and *who* were used. Embedding the richer information coming from the event's *attributes* can potentially allow for a more precise definition of the player's strategy. An option could be assigning weights to these values as well while carefully distinguish between categorical, ordinal and numerical attributes.

Since EFs could define and express more focused strategies, several behavioural metrics could be defined and used to numerically evaluate the differences between the agents presented so far.

Finally, the Event-value Functions are able to represent in a very compact representation a potentially very expressive behavioural space. Such space could be explored using a MAP-Elites algorithm that has shown very promising results [25] in such tasks. The behavioural space would be described by metrics as the ones mentioned above.

ACKNOWLEDGEMENTS

This work was funded by the EPSRC CDT in Intelligent Games and Game Intelligence (IGGI) EP/L015846/1. This research utilised Queen Mary's Apocrita HPC facility [26], supported by QMUL Research-IT.

REFERENCES

[1] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of Monte Carlo Tree Search Methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[2] D. Perez, S. Samothrakis, S. Lucas, and P. Rohlfshagen, "Rolling Horizon Evolution versus Tree Search for navigation in single-player real-time games," in *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. ACM, 2013, pp. 351–358.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari With Deep Reinforcement Learning," in *NIPS Deep Learning Workshop*, 2013.

[4] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[5] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.

[6] D. Perez-Liebana, J. Liu, A. Khalifa, R. D. Gaina, J. Togelius, and S. M. Lucas, "General Video Game AI: A Multitrack Framework for Evaluating Agents, Games, and Content Generation Algorithms," *IEEE Transactions on Games*, vol. 11, no. 3, pp. 195–214, 2019.

[7] I. Bravi and S. M. Lucas, "Rinascimento: using event-value functions for playing Splendor," in *2020 IEEE Conference on Games (CoG)*. IEEE, 2020, pp. 283–290.

[8] T. Pohlen, B. Piot, T. Hester, M. G. Azar, D. Horgan, D. Budden, G. Barth-Maron, H. Van Hasselt, J. Quan, M. Večerík *et al.*, "Observe and look further: Achieving consistent performance on atari," *arXiv preprint arXiv:1805.11593*, 2018.

[9] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune, "First return, then explore," *Nature*, vol. 590, no. 7847, pp. 580–586, 2021.

[10] Y. Aytar, T. Pfaff, D. Budden, T. Paine, Z. Wang, and N. de Freitas, "Playing hard exploration games by watching YouTube," in *Advances in Neural Information Processing Systems*, 2018, pp. 2930–2941.

[11] T. Salimans and R. Chen, "Learning Montezuma's Revenge from a Single Demonstration," *arXiv preprint arXiv:1812.03381*, 2018.

[12] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Icml*, vol. 99, 1999, pp. 278–287.

[13] Y. Hu, W. Wang, H. Jia, Y. Wang, Y. Chen, J. Hao, F. Wu, and C. Fan, "Learning to utilize shaping rewards: A new approach of reward shaping," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[14] C. Holmgård, M. C. Green, A. Liapis, and J. Togelius, "Automated playtesting with procedural personas through mcts with evolved heuristics," *IEEE Transactions on Games*, vol. 11, no. 4, pp. 352–362, 2018.

[15] D. Perez, S. Samothrakis, and S. Lucas, "Knowledge-based fast evolutionary MCTS for general video game playing," in *2014 IEEE Conference on Computational Intelligence and Games*. IEEE, 2014, pp. 1–8.

[16] A. Barreto, W. Dabney, R. Munos, J. J. Hunt, T. Schaul, H. P. van Hasselt, and D. Silver, "Successor features for transfer in reinforcement learning," in *Advances in neural information processing systems*, 2017, pp. 4055–4065.

[17] P. Dayan, "Improving generalization for temporal difference learning: The successor representation," *Neural Computation*, vol. 5, no. 4, pp. 613–624, 1993.

[18] F. de Mesentier Silva, R. Canaan, S. Lee, M. C. Fontaine, J. Togelius, and A. K. Hoover, "Evolving the hearthstone meta," in *2019 IEEE Conference on Games (CoG)*. IEEE, 2019, pp. 1–8.

[19] I. Bravi, D. Perez-Liebana, S. M. Lucas, and J. Liu, "Rinascimento: Optimising Statistical Forward Planning Agents for Playing Splendor," in *2019 IEEE Conference on Games (CoG)*. IEEE, 2019, pp. 1–8.

[20] H. Noguchi, R. Ishii, T. Harada, and R. Thawonmas, "Improving rolling horizon evolutionary algorithm in a fighting game," in *2019 Nicograph International (NicoInt)*. IEEE, 2019, pp. 118–118.

[21] S. M. Lucas, J. Liu, I. Bravi, R. D. Gaina, J. Woodward, V. Volz, and D. Perez-Liebana, "Efficient evolutionary methods for game agent optimisation: Model-based is best," *arXiv preprint arXiv:1901.00723*, 2019.

[22] S. M. Lucas, J. Liu, and D. Perez-Liebana, "The n-tuple bandit evolutionary algorithm for game agent optimisation," in *2018 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2018, pp. 1–9.

[23] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[24] M. Zhu and S. Gupta, "To Prune, or Not to Prune: Exploring the Efficacy of Pruning for Model Compression," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*, 2018.

[25] J.-B. Mouret and J. Clune, "Illuminating search spaces by mapping elites," *arXiv preprint arXiv:1504.04909*, 2015.

[26] T. King, S. Butcher, and L. Zalewski, *Apocrita - High Performance Computing Cluster for Queen Mary University of London*, Mar. 2017. [Online]. Available: https://doi.org/10.5281/zenodo.438045