Towards Action Model Learning for Player Modeling

Anonymous 1
email1@email.email

Anonymous 2
email2@email.email

Address 1
Address 2
City, Country

Abstract

Player modeling attempts to create a computational model which accurately approximates a player's behavior in a game. Most player modeling techniques rely on domain knowledge and are not transferable across games. Additionally, player models do not currently yield any explanatory insight about a player's cognitive processes, such as the creation and refinement of mental models. In this paper, we present our findings with using *action model learning* (AML), in which an action model is learned given data in the form of a play trace, to learn a player model in a domain-agnostic manner. We demonstrate the utility of this model by introducing a technique to quantitatively estimate how well a player understands the mechanics of a game. We evaluate an existing AML algorithm (FAMA) for player modeling and develop a novel algorithm called Blackout that is inspired by player cognition. We compare Blackout with FAMA using the puzzle game Sokoban and show that Blackout generates better player models.

Introduction

Player modeling is the study of computational models of players in games (Yannakakis et al. 2013). It sees varied and widespread usage in today's video game industry. Imangi Studios, developers of the popular *Temple Run* (Imangi Studios 2011) series of mobile games, collect player telemetry in order to analyse player behavior and provide customized gameplay experiences. *Forza Motorsport 5* (Turn 10 Studios 2013) implements a *Drivatar* system which learns to mimic the player's behavior in the game and simulate the player in other races.

Despite the multitude of techniques used to build player models, such as self-organizing maps (Drachen, Canossa, and Yannakakis 2009), Bayesian networks (Yannakakis and Maragoudakis 2005), and multi-layer perceptrons (Pedersen, Togelius, and Yannakakis 2010), many of them rely on features extracted from domain knowledge of the game's rules and as such cannot be generalized easily, except perhaps to games of the same genre. The inability to easily train new models for different games using the same technique presents a barrier to any single technique's adoption.

Furthermore, while current techniques aim to *predict* player actions, we argue that there is a corresponding need to be *explain* their underlying cognitive processes. There is empirical evidence in cognitive science that game players build *mental models*, knowledge structures capable of simulating the system they are interacting with and predicting and explaining the outcomes of scenarios, while playing games (Boyan, McGloin, and Wasserman 2018). Mental models may start out erroneous, but improve over time as the player learns from which of their attempted actions succeed and fail. We believe that mental model alignment is one of the most important aspects of using games for impactful applications, such as education and training, and that player modeling techniques should be designed to yield insight into this cognitive process.

To address these needs, we propose *action model learning* (AML), in which an action model is learned from play traces, as a viable technique. Action models can be used to learn player models in any game which can be represented in a planning formalism like PDDL. There is a rich body of literature on learning action models from action traces which we can leverage to learn action models. In this paper, we describe our efforts to build an action model-based player model which can be used to characterize player competency. To ascertain the feasibility of action model learning as an approach to player modeling, we started with a modern algorithm known as FAMA (Aineto, Celorrio, and Onaindia 2019) out-of-the-box. In parallel, we developed an in-house alternative that we call Blackout, motivated by mental model alignment and taking failed actions into account. We test both approaches on the puzzle game Sokoban and evaluate their output, finding that Blackout outperforms FAMA for the task of player modeling. We discuss both techniques' advantages and limitations and suggest avenues for future work.

While this paper's quantitative results focus on the comparison between AML and Blackout, more generally, we argue that AML is a *tractable* and *domain-agnostic* approach to player modeling, and that the learned action model is a *useful* player model. We justify the tractability claim by successfully applying both FAMA and Blackout to the task of learning a player model from action traces, measuring their

efficiency on problems of various sizes as well as precision and recall. We justify the domain-agnosticity claim by successfully learning action models for two additional domains: Hanoi (the Tower of Hanoi puzzle) and N-puzzle (a sliding tile puzzle), using publicly available domain files[1] and trajectories from FAMA's evaluation dataset. We justify the usefulness claim by presenting a technique to quantify a player's mechanical mastery of a game given their action model-based player model.

Related Work

Player modeling is a relatively new field, previously studied under the umbrella of HCI in the form of user modeling (Biswas and Springett 2018) and student modeling. (Chrysafiadi and Virvou 2013) Recent surveys on player modeling provide useful taxonomies (Smith et al. 2011), an overview of techniques that have been used for player modeling generally (Machado, Fantini, and Chaimowicz 2011) or for MMORPGs (Harrison and Roberts 2011), and challenges commonly encountered in the field, including the reliance on knowledge engineering (Hooshyar, Yousefi, and Lim 2018) that motivates our work.

Domain-agnostic approaches to player modeling have been attempted before. Snodgrass, Mohaddesi, and Harteveld (2019) describe a general player model using the PEAS framework, which presents a theoretical framework for providing game recommendations, but does not account for in-game player behavior. Nogueira et al. (2014) use physiological readings of players to model their emotional response to game events. While input-agnostic, this technique relies on physiological data from sensors, which is difficult to acquire. Our approach requires only gameplay traces, which can be easily done by making minor modifications to the game engine code. Deep neural networks have been used to simulate the actions of human players in interactive fiction games (Wang et al. 2018) and MMORPGs (Pfau, Smeddinck, and Malaka 2018) and to generate new levels in a platformer game based on learned player preferences (Summerville et al. 2016). These techniques rely only on easily obtainable input like gameplay logs or video recordings, and do not use any knowledge engineering to train the model. We believe our approach has two advantages: that it does not require as much data to learn, and, by generating a rule-based model, offers more explanatory power for player behavior.

To the best of our knowledge, this is the first-known application of AML to player modeling. The literature on AML has primarily focused on learning sound action models for use by an automated planner, with little attention paid to modeling the player's cognitive processes. *Human-aware planning* attempts to create planning agents which can take human mental models into account while planning (Chakraborti et al. 2017), while we attempt to treat a human as a planning agent and learn a domain model which mirrors their mental model. Serafini and Traverso (2019) introduce a perception function which maps sensor data to state variables in order to learn action models. We believe investigating player perception functions for cases like player disability to learn action models would be fruitful to pursue. An early AML algorithm called OBSERVER uses action failures to learn action preconditions (Wang 1995), which mirrors our approach.

[1] http://planning.domains/

Background

Action model learning (AML) is situated within the tradition of automated planning, in which an *action model* describes the preconditions and effects of each action an agent may take in a world. Preconditions and effects are propositional formulas over fluents (predicates whose truth changes with time). A *plan* is a sequence of actions, each instantiated with terms for each parameter position, such that the effects of each action entail the preconditions of the action following it. AML, then, is the problem of discovering an action model from a set of observed plans.

More formally, an AML algorithm takes as input a number of *action traces*, each a sequence $\tau = \langle s_0, a_1, s_1, a_2, s_2, \cdots, a_n, s_n \rangle$, where s_i are states and a_i are actions, and returns as output an action model, which is a specification of the preconditions and effects of every action in the domain. These action traces are usually assumed to be fully observed, i.e. every fluent of a state is present, but various developments in AML algorithms allow action models to be learned from action traces with partially observed or noisy states as well. An action model is represented as a STRIPS-style domain, with the most common representation format being PDDL.

The primary domain used in development and testing is the classic Japanese block-pushing puzzle game Sokoban, which we use to demonstrate the feasibility of our approach. We select it based on its relative minimalism and existing formalizations in PDDL as part of the International Planning Competition (IPC) benchmark suite (Coles et al. 2012). In Sokoban, there are stones (or crates) scattered around a warehouse floor, which the player must push to certain specified locations around the warehouse, but can only push the stones in cardinal directions, cannot climb over the stones or other obstacles, and can only push one stone at a time. For an overview of Sokoban's rules and its characteristics as a search problem, we refer readers to Junghanns and Schaeffer (1997). The specific representation of Sokoban we use is a domain that appeared in the 2011 International Planning Competition,[2] modified to remove action costs.

Player Modeling with FAMA

To determine baseline feasibility of action model learning as an approach to player modeling, we started with an out-of-the-box algorithm known as FAMA (Aineto, Celorrio, and Onaindia 2019). FAMA performs AML by compiling the learning task into a classical planning task. This is done by creating actions in the compiled planning domain which correspond to inserting fluents in the preconditions or effects of various actions in the planning domain to be learned. The

[2] https://github.com/potassco/pddl-instances/blob/master/ipc-2011/domains/sokoban-sequential-satisficing/domain.pddl

```
(:action move
  :parameters
    (?p - player
     ?from -
    location
     ?to - location
     ?dir -
    direction)
  :precondition (
    and
     (clear ?to)
     (at ?p ?from)
     (is-nongoal ?
    from))
  :effect (and
     (clear ?from)
     (at ?p ?to)
     (move-dir ?from
      ?to ?dir)
     (not (at ?p ?
    from))
     (not (clear ?to
    ))))
```

```
(:action move
  :parameters
    (?p - player
     ?from -
    location
     ?to - location
     ?dir -
    direction)
  :precondition (
    and
     (clear ?to)
     (at ?p ?from)
     (move-dir ?from
      ?to ?dir))
  :effect (and
     (clear ?from)
     (at ?p ?to)
     (is-nongoal ?to
    )
     (is-nongoal ?
    from)
     (not (at ?p ?
    from))
     (not (clear ?to
    ))))
```

Figure 1: A comparison between the `move` action learned by M_1 and M_2

solution plan is thus a sequence of actions which builds up the output domain model and verifies its consistency.

FAMA has properties which are useful for our problem. It is capable of working with action traces having incomplete states (states with missing fluents) and missing actions. Since it utilizes classical planning, we can use a variety of existing planners in our system based on their properties such as computation time and memory availability. The action models it produces are formally sound and can be used to generate new trajectories. It is capable of learning action models from multiple different trajectories.

In lieu of a human player, we use the FastDownward 19.12 planning system to generate a solution path for each of two custom levels, then convert each solution into an action trace which is input to FAMA. We use FAMA with full state observability to successfully learn two action models M_1 and M_2 from each of the two obtained action traces. Figure 1 shows the `move` action learned by both models.

Player Modeling with Blackout

Our new AML algorithm, Blackout, takes advantage of stronger assumptions that can be made of the input (such as full observability) to improve execution speed and model accuracy. Blackout also takes action *failures* into account based on the cognitive theory that failed actions inform players' mental models.

The inputs to Blackout are (1) a play trace and (2) a domain file $D = \{\Theta, \Psi, \alpha\}$, where Θ is the set of types, Ψ is the set of predicates, and α is the action model. The action model consists of actions with positive and negative preconditions and effects represented by $\texttt{pre}^{+}(a)$,$\texttt{pre}^{-}(a)$,$\texttt{eff}^{+}(a)$ and $\texttt{eff}^{-}(a)$ respectively. We assume deterministic action effects with full observability of states and actions and no noise. We believe this is a reasonable assumption given how the trajectories are obtained. Blackout outputs a STRIPS action model which is not guaranteed to be sound.

Blackout operates in three steps: first, it produces an initial action model by analyzing the differences between the *pre-state* and *post-state* for every action in the trajectory. Next, it uses information from failed action executions to improve the initial action model. Finally, it computes invariants which hold for predicates in the domain to further improve the action model. We now explain each step in detail.

Step 1: Successful Action Analysis

We first compute the effects of actions by calculating the *delta state* for each action execution. For discovering preconditions, we first calculate the set of all possible fluents which could be valid preconditions given the domain and the action's bindings. This is done by binding all possible combinations of objects already bound to an action's arguments, given by $\texttt{obj}(a)$, to all predicates in the domain. Assignment of a set of objects ω to a predicate p is represented by $p(\omega)$, assuming $\|\omega\| = \texttt{arity}(p)$ and there is no type mismatch. Applicable preconditions are first *generalized* in the function G i.e. their bindings are replaced with variables and then added to the precondition lists. The action model output in this step may contain superfluous and erroneous preconditions.

Algorithm 1: Successful Action Analysis

Data: a set T of trajectories of the form $\langle s_0, a_0, s_1, a_1, s_2, \cdots, a_{n-1}, s_n \rangle$, a reference domain model $D = \{\Theta, \Psi, \alpha\}$
Result: a STRIPS action model A
begin
  Initialize empty action model A;
  foreach *trajectory* $t \in T$ **do**
    foreach $\langle s - a - s' \rangle$ *in* t **do**
      $\overline{s} \leftarrow s \setminus \{p \mid p \in s, \texttt{obj}(p) \notin \texttt{obj}(a)\}$;
      $\overline{s'} \leftarrow s' \setminus \{p \mid p \in s', \texttt{obj}(p) \notin \texttt{obj}(a)\}$;
      $F \leftarrow \{p(\omega) \mid p \in \Psi, \omega \in \texttt{obj}(a)^{\texttt{arity}(p)}\}$;
      $\texttt{pre}^{+}(a) \leftarrow G(F \setminus \overline{s})$;
      $\texttt{pre}^{-}(a) \leftarrow G(F \setminus \overline{s'})$;
      $\texttt{eff}^{+}(a) \leftarrow G(\overline{s'} \setminus \overline{s})$;
      $\texttt{eff}^{-}(a) \leftarrow G(\overline{s} \setminus \overline{s'})$;
    end
  end
  return A
end

Step 2: Failed Action Analysis

Failed actions are actions which the player attempts to perform in-game, but which cannot be completed due to the preconditions of an action not being met. It indicates a mismatch between the player's mental model of the game's mechanics and the actual action model representing the game.

Blackout makes the assumption that players notice the failure of action execution and update their mental model to account for this. We record failed actions in a manner similar to successful actions in the trajectory, with failed actions forming triplets of the form $\langle s - a_f - s \rangle$. The pre-state and post-state are identical since the action failed to execute. Blackout attempts to identify the predicates in the pre-state which caused the action a_f to fail to execute due to violating its preconditions. These are stored in the sets $R^+_{a_f}$ and $R^-_{a_f}$ for positive and negative preconditions respectively. The detailed algorithm is described in Algorithm 2.

Algorithm 2: Failed Action Analysis

Data: a set T of trajectories of the form $\langle s_0, a_0, s_1, a_1, s_2, \cdots, a_{n-1}, s_n \rangle$ with failed actions a_f, a reference domain model $D = \{\Theta, \Psi, A\}$, action model A from step 1
Result: a STRIPS action model A
begin
  foreach *trajectory* $t \in T$ **do**
    foreach $\langle s - a_f - s \rangle$ *in* t **do**
      $R^+_{a_f} \leftarrow R^+_{a_f} \cup (\texttt{pre}^+(a_f) \setminus s)$;
      $R^-_{a_f} \leftarrow R^-_{a_f} \cup (\texttt{pre}^-(a_f) \cap s)$;
      if $R^+_{a_f}$ *is a singleton set* & $R^-_{a_f} = \emptyset$ **then**
        Mark $p \in R^+_{a_f}$ as confirmed;
      else if $R^+_{a_f} = \emptyset$ & $R^-_{a_f}$ *is a singleton set* **then**
        Mark $p \in R^-_{a_f}$ as confirmed;
      else
        Mark $\langle R^+_{a_f}, R^-_{a_f}, a_f \rangle$ as ambiguous;
      end
    end
  end
  foreach *action* $a \in A$ **do**
    $\texttt{pre}^+(a) \leftarrow G(\{p \mid p \in \texttt{pre}^+(a)$ where p is confirmed$\})$;
    $\texttt{pre}^-(a) \leftarrow G(\{p \mid p \in \texttt{pre}^-(a)$ where p is confirmed$\})$;
  end
  return A
end

Step 3: Invariant Extraction

In this step, Blackout identifies invariants that hold among pairs of predicates in the domain. An invariant between two predicates p and q for some shared variable v is a relation on $\{(p,q),(p,\neg q),(\neg p,q),(\neg p,\neg q)\}\} \times \mathbb{B}$. It describes what combinations of p and q are permitted to be attached to v or not (attachment denoted by p and q, lack thereof by $\neg p$ and $\neg q$) after an action has been performed on v. Blackout uses these invariants to resolve any ambiguities regarding the predicates responsible for action failure in step 2, and in doing so further refine the action preconditions.

Blackout first tries to identify invariants in the action effects. It does so by looking for *candidate primitive rules*, relations between predicates p and q asserting that q's addition implies p's addition (or removal, depending on whether p and q are in $\texttt{eff}^+$ or $\texttt{eff}^-$). Candidate primitive rules are merged where possible to form *proper primitive rules*. If these proper primitive rules cover the four cases for p and q's addition and removal, they represent an invariant.

However, the invariants identified so far aren't necessarily true for the initial state of the trajectory. For instance, in the IPC problem files, the "wall" cells each get a `location` object with neither an `at` predicate nor a `clear` predicate attached. This breaks the mutual exclusivity invariant we observe when looking at just the action effects. In addition, some predicates (namely `MOVE-DIR`, `IS-GOAL`, and `IS-NONGOAL`) never appear in the action effects at all, which means the above process is entirely blind to them, even though these predicates can encode useful information. To solve both these issues, Blackout analyzes the initial state of the system s_0 when given a trajectory t.

Blackout includes a procedure $\texttt{Invariant}(p, q, v, s_0)$, which returns candidate invariant relations between p and q that hold in the initial state. These invariants are unioned with those from action effects to form the final list of invariants in the domain. These invariants are guaranteed to hold in any state reachable from the initial state, provided that the effects discovered in step 1 are accurate.

These invariants are used to resolve ambiguous failed actions from step 2. If any invariant refers to two different predicates and their corresponding arguments in the same R_{a_f}, Blackout takes one of five possible actions on the $\texttt{pre}(a_f)$ sets based on the invariant type. We document these in Table 2.

Evaluation

Aineto, Celorrio, and Onaindia (2019) introduced a novel metric for evaluating action models in the form of precision and recall, which we adapt to our task of producing a score which measures a player's understanding of a particular mechanic. This technique is a slight modification of their evaluation scheme.

We make the assumption that every action in the domain of a game defined in PDDL corresponds to a mechanic. We compare the learned action model to the ground truth model and *for each action* count the number of predicates in the learned model's preconditions and effects which are correct (true positives), extra (false positives) and missing (false negatives). We use this confusion matrix to compute the F_1-score for every action in the model and report it as the player's proficiency score for a particular mechanic given the model, as shown in Table 3. We use the F_1-score since it is a meaningful way to combine precision and recall into a

Algorithm 3: Invariant Extraction

Data: a set T of trajectories of the form $\langle s_0, a_0, s_1, a_1, s_2, \cdots, a_{n-1}, s_n \rangle$ with failed actions a_f, a reference domain model $D = \{\Theta, \Psi, \alpha\}$, action model A from step 2, $R^+_{a_f}$ and $R^-_{a_f}$ from step 2
Result: a STRIPS action model A
begin
  $P \leftarrow \emptyset$;
  foreach *action* $a \in A$ **do**
    foreach $(p_1, p_2) \in (\texttt{eff}^+(a) \cup \texttt{eff}^-(a))^2, p_1 \neq p_2$ **do**
      foreach *variable* $v \in \texttt{var}(p_1) \cap \texttt{var}(p_2)$ **do**
        $P \leftarrow P \cup$ PrimitiveRule(p_1, p_2, v);
      end
    end
  end
  foreach *primitive rule* $r(p, q, v) \in P$ **do**
    if $\exists a \in A, q \in \texttt{eff}^+(a) \cup \texttt{eff}^-(a), p \notin \texttt{eff}^+(a) \cup \texttt{eff}^-(a)$ **then**
      $P \leftarrow P \setminus \{r\}$;
    end
  end
  $I \leftarrow \emptyset$;
  foreach *primitive rule* $r(p, q, v) \in P$ **do**
    $M \leftarrow \{r\}$;
    foreach *primitive rule* $r'(p', q', v') \in P, r' \neq r$ **do**
      if $v = v' \wedge ((p = p' \wedge q = q') \vee (p = q' \wedge q = p'))$ **then**
        $M \leftarrow M \cup \{r'\}$;
      end
    end
    if $|M| = 4$ **then**
      $I \leftarrow I \cup \{\texttt{Invariant}(M)\}$;
    end
  end
  $J \leftarrow \emptyset$;
  foreach *trajectory* $t \in T$ **do**
    foreach *predicate* $p \in \Psi$, *predicate* $q \in \Psi$, *variable* $v \in \texttt{var}(p_1) \cap \texttt{var}(p_2), p \neq q$ **do**
      $J \leftarrow J \cup \{\texttt{Invariant}(p, q, v, s_{0,t})\}$;
    end
  end
  $K \leftarrow \texttt{Merge}(I, J)$;
  foreach *ambiguous* $\langle R^+_{a_f}, R^-_{a_f}, a_f \rangle$ **do**
    if $\exists k(p, q, v, op) \in K, \{p, q\} \subseteq R^+_{a_f} \cup R^-_{a_f}$ **then**
      if $op = \oplus \vee op = \odot$ **then**
        Mark p and q as confirmed;
      end
    end
  end
  return A;
end

Invariant (pRq)	Meaning
$\bot$	Neither predicate can ever be present, and neither can ever be absent
$p \wedge q$	Both predicates must always be present
$p \not\Longleftarrow q$	First predicate is always present, second is always absent
p	First predicate is always present, second may be present or absent
$p \not\Longrightarrow q$	First predicate is always absent, second is always present
q	Second predicate is always present, first may be present or absent
$p \oplus q$	Predicates are mutually exclusive
$p \vee q$	At least one of the predicates must always be present
$p \downarrow q$	Neither predicate can ever be present
$p \odot q$	Predicates are equivalent: when one is present, the other is too
$\neg q$	Second predicate is always absent, first can be present or absent
$p \Longleftarrow q$	If the second predicate is present, the first must also be present
$\neg p$	First predicate is always absent, second can be present or absent
$p \Longrightarrow q$	If the first predicate is present, the second must also be present
$p \uparrow q$	At most one of the predicates can be present
$\top$	Predicates are independent. Each can be present or absent, regardless of the other

Table 1: List of all possible invariants and their meanings

single number.

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{1}$$

To compare FAMA and Blackout, we ran them both on traces generated manually from two hand-crafted levels (L_1 and L_2) and an instance from the IPC 2011 collection (L_3). The levels differ in complexity, with L_1 being the simplest and L_3 being the most complex. We modify the evaluation scheme proposed by Aineto, Celorrio, and Onaindia (2019) to measure precision and recall for *each action* for each level. Since recall was found to be constant across all Blackout stages and FAMA, for all levels, differing only by ac-

Invariants	Action
$\bot, \wedge, \not\Longleftarrow, \not\Longrightarrow, \downarrow$	Error
$p, \neg p$	q is confirmed as a precondition
$q, \neg q$	p is confirmed as a precondition
$\vee. \Longrightarrow, \Longleftarrow, \uparrow, \top$	No action
$\oplus, \odot$	p and q are confirmed as preconditions

Table 2: The actions on the R_{a_f} sets when a matching invariant is found

Mechanic name	F_1-score	
	M_1	M_2
move	0.267	0.250
push-to-nongoal	0.308	0.216
push-to-goal	0.222	0.240

Table 3: Proficiency scores for each mechanic

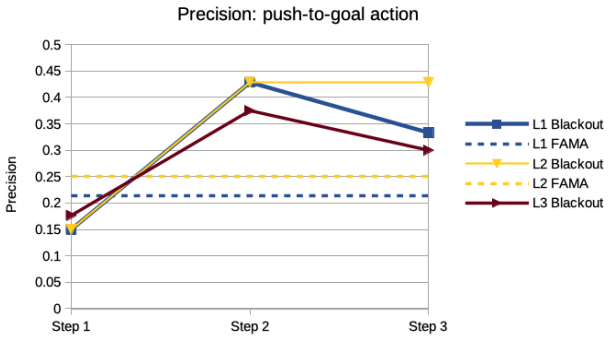

Figure 2: Precision scores for Blackout and FAMA for the push-to-goal action

tions, we report the recall values separately in Table 4. Our measurements show that Blackout achieves better precision across all actions while achieving the same recall.

We do not report FAMA metrics for L_3 since we were unable to obtain an action model from it in a reasonable time. We also do not report results for Blackout on `push-to-nongoal` since it doesn't appear in the trajectory for that level and so Blackout does not know that it exists. We display the precision scores for Blackout and FAMA for the `push-to-goal` action in Figure 2. The other actions have similar trends, so we chose not to report them.

We notice that Step 3 of Blackout tends to decrease the precision scores in comparison to Step 2, as shown in Figure 2. Whether this is indicative of a flaw in the algorithm or in the metric will be investigated in future work.

Performance Scalability

We use a machine with an Intel Core i7-9750H CPU with 16 GB RAM to compare the elapsed time and memory usage of FAMA and Blackout in learning action models. We attempt to learn a single action model for each of L_1, L_2 and L_3. To make our trajectories compatible with FAMA, we strip them of failed actions. We measure elapsed time and memory usage via *wall clock time* and *maximum resident set size* as output by the Unix `time` utility. We repeat each run of the

Action	Recall
`move`	0.2857
`push-to-nongoal`	0.3077
`push-to-goal`	0.2308

Table 4: Recall values for each action across all Blackout stages and FAMA, for levels L_1, L_2 and L_3

		Blackout			FAMA
		S_1	S_{1+2}	S_{1+2+3}	
L_1	Time (s)	0.11	0.10	0.11	10.59
	Mem. (MB)	6.96	6.90	7.08	1572.60
L_2	Time (s)	0.10	0.09	0.11	21.19
	Mem. (MB)	6.80	6.78	6.99	4632.52
L_3	Time (s)	2.31	2.34	2.48	—
	Mem. (MB)	35.82	35.78	35.82	—

Table 5: Performance comparison between FAMA and Blackout for player modeling

algorithm for each trajectory 30 times and report the mean for each measured metric. Our findings for FAMA and for each step of Blackout are listed in Table 5.

FAMA was unsuccessful in learning action models from a relatively large trajectory file ($\sim$ 1 MB); however, we expect player trajectories to be an order of magnitude larger in size ($>$ 1 GB). We see that Blackout to be much faster in learning action models than FAMA while having a smaller memory footprint. However, further work needs to be done to assess performance in the case of more realistic trajectory sizes.

All of our code, including the tests, data, and Blackout implementation, is available in an open-source repository (omitted for anonymous review).

Conclusion and Future Work

To the best of our knowledge, our work is the first application of AML to player modeling. We demonstrated the feasibility of AML as a domain-agnostic player modeling approach by evaluating an existing AML algorithm for the task of player modeling, and we presented a technique to measure the player's mechanical proficiency using the learned model. We also introduced Blackout, a novel AML algorithm that incorporates action failures, compared the two approaches, and found that Blackout learns action models better and faster than FAMA. Based on these findings, we believe action model learning has the potential to serve as a useful player modeling technique in a wide range of games.

We were motivated in this project by a hypothesis that the cognitive process of mental model formation (Wasserman and Koban 2019) could help design better AML algorithms for player modeling. Blackout is informed by this motivation in updating action models based on failed actions, which are used similarly in mental model alignment (Boyan, McGloin, and Wasserman 2018). However, we have not yet conducted an experiment to measure how well our learned models match human players' mental models; we plan to do this in future work.

Our proposed measure of player competency will also benefit from refinement in future work. Quantitatively estimating a player's mechanical proficiency leaves out higher-order skills learned by combining the use of mechanics that players need to learn in order to become proficient at a game (Cook 2007). Future work will attempt to model and measure the acquisition of higher-order skills.

References

Aineto, D.; Celorrio, S. J.; and Onaindia, E. 2019. Learning action models with minimal observability. *Artificial Intelligence* 275:104 – 137.

Biswas, P., and Springett, M. 2018. User modeling. *The Wiley Handbook of Human Computer Interaction Volume* 143.

Boyan, A.; McGloin, R.; and Wasserman, J. A. 2018. Model matching theory: A framework for examining the alignment between game mechanics and mental models. *Media and Communication* 6(2):126–136.

Chakraborti, T.; Kambhampati, S.; Scheutz, M.; and Zhang, Y. 2017. AI challenges in human-robot cognitive teaming. *CoRR* abs/1707.04775.

Chrysafiadi, K., and Virvou, M. 2013. Student modeling approaches: A literature review for the last decade. *Expert Systems with Applications* 40(11):4715 – 4729.

Coles, A.; Coles, A.; Olaya, A. G.; Jiménez, S.; López, C. L.; Sanner, S.; and Yoon, S. 2012. A survey of the seventh international planning competition. *AI Magazine* 33(1):83–88. Copyright - Copyright Association for the Advancement of Artificial Intelligence Spring 2012; Document feature - ; Diagrams; Last updated - 2018-10-06.

Cook, D. 2007. The chemistry of game design. Accessed May 15, 2020 from https://www.gamasutra.com/view/feature/129948/the_chemistry_of_game_design.php.

Drachen, A.; Canossa, A.; and Yannakakis, G. N. 2009. Player modeling using self-organization in tomb raider: Underworld. In *2009 IEEE Symposium on Computational Intelligence and Games*, 1–8.

Harrison, B., and Roberts, D. L. 2011. Using sequential observations to model and predict player behavior. In *Proceedings of the 6th International Conference on Foundations of Digital Games*, 91–98.

Hooshyar, D.; Yousefi, M.; and Lim, H. 2018. Data-driven approaches to game player modeling: A systematic literature review. *ACM Comput. Surv.* 50(6).

Junghanns, A., and Schaeffer, J. 1997. Sokoban: A challenging single-agent search problem. In *In IJCAI Workshop on Using Games as an Experimental Testbed for AI Reasearch*. Citeseer.

Machado, M. C.; Fantini, E. P.; and Chaimowicz, L. 2011. Player modeling: Towards a common taxonomy. In *2011 16th international conference on computer games (CGAMES)*, 50–57. IEEE.

Nogueira, P. A.; Aguiar, R.; Rodrigues, R. A.; Oliveira, E. C.; and Nacke, L. 2014. Fuzzy affective player models: A physiology-based hierarchical clustering method. In *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Pedersen, C.; Togelius, J.; and Yannakakis, G. N. 2010. Modeling player experience for content creation. *IEEE Transactions on Computational Intelligence and AI in Games* 2(1):54–67.

Pfau, J.; Smeddinck, J. D.; and Malaka, R. 2018. Towards deep player behavior models in mmorpgs. In *Proceedings of the 2018 Annual Symposium on Computer-Human Interaction in Play*, CHI PLAY '18, 381–392. New York, NY, USA: Association for Computing Machinery.

Serafini, L., and Traverso, P. 2019. Incremental learning of discrete planning domains from continuous perceptions. *arXiv preprint arXiv:1903.05937*.

Smith, A. M.; Lewis, C.; Hullet, K.; Smith, G.; and Sullivan, A. 2011. An inclusive view of player modeling. In *Proceedings of the 6th International Conference on Foundations of Digital Games*, 301–303.

Snodgrass, S.; Mohaddesi, O.; and Harteveld, C. 2019. Towards a generalized player model through the peas framework. In *Proceedings of the 14th International Conference on the Foundations of Digital Games*, FDG '19. New York, NY, USA: Association for Computing Machinery.

Summerville, A.; Guzdial, M.; Mateas, M.; and Riedl, M. O. 2016. Learning player tailored content from observation: Platformer level generation from video traces using lstms. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Wang, P.; Rowe, J.; Min, W.; Mott, B.; and Lester, J. 2018. High-fidelity simulated players for interactive narrative planning. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, 3884–3890. AAAI Press.

Wang, X. 1995. Learning by observation and practice: An incremental approach for planning operator acquisition. In *Machine Learning Proceedings 1995*. Elsevier. 549–557.

Wasserman, J. A., and Koban, K. 2019. Bugs on the brain: A mental model matching approach to cognitive skill acquisition in a strategy game. *Journal of Expertise/June* 2(2).

Yannakakis, G. N., and Maragoudakis, M. 2005. Player modeling impact on player's entertainment in computer games. In Ardissono, L.; Brna, P.; and Mitrovic, A., eds., *User Modeling 2005*, 74–78. Berlin, Heidelberg: Springer Berlin Heidelberg.

Yannakakis, G. N.; Spronck, P.; Loiacono, D.; and André, E. 2013. Player modeling.