HRLB^2: A Reinforcement Learning Based Framework for Believable Bots

Christian Arzate Cruz * and Jorge Adolfo Ramirez Uresti *

School of Engineering and Science, Tecnologico de Monterrey, CP 64849 Monterrey, Mexico
* Correspondence: arzate.christian@gmail.com (C.A.C.); juresti@tec.mx (J.A.R.U.)

Received: 24 October 2018; Accepted: 20 November 2018; Published: 1 December 2018

Abstract: The creation of believable behaviors for Non-Player Characters (NPCs) is key to improve the players’ experience while playing a game. To achieve this objective, we need to design NPCs that appear to be controlled by a human player. In this paper, we propose a hierarchical reinforcement learning framework for believable bots (HRLB^2). This novel approach has been designed so it can overcome two main challenges currently faced in the creation of human-like NPCs. The first difficulty is exploring domains with high-dimensional state–action spaces, while satisfying constraints imposed by traits that characterize human-like behavior. The second problem is generating behavior diversity, by also adapting to the opponent’s playing style. We evaluated the effectiveness of our framework in the domain of the 2D fighting game named Street Fighter IV. The results of our tests demonstrate that our bot behaves in a human-like manner.

Keywords: Game AI; human-like behavior; believable bot; behavior imitation; reinforcement learning

1. Introduction

In recent years, the Game AI community has made many efforts to accomplish a better understanding of how Theory of Flow constructs would be essential to improve current approaches in the player-centered subarea [1]. We can argue that the main reasons to study flow in the context of Game AI are the effects of achieving this state of optimal experience; that is, people enjoy the most when achieving this subjective state of consciousness [2]. In more common terms, we can define flow as a lasting and deep state of immersion [1].

Therefore, creating more immersive experiences is key to enhancing players’ experience while playing a game. An approach to meet this goal is generating believable behaviors for Non-Player Characters (NPCs) [1]. A believable NPC behaves in a manner that makes it indistinguishable from human players. Therefore, to approach the design of believable NPCs, we need to identify which traits characterize a human-like behavior [3,4], and how those traits can be achieved through artificial intelligence techniques (AI) [1,5–10].

Reinforcement Learning (RL) is a popular technique that is effective in learning how to play a wide range of games, such as chess [11] or First Person Shooters (FPSs) [12]. Furthermore, a RL approach has even been able to defeat world-class Go players [13]. However, the use of RL to create believable bots has been limited. In this paper, we propose a model-based hierarchical reinforcement learning framework for believable bots—HRLB^2. With this novel application of RL emerged two main challenges.

The first challenge we found is exploring domains with high-dimensional state–action spaces, while satisfying constraints imposed by traits that characterize human-like behavior. To approach this problem, our framework learns the model of a game by observing how humans play that game. The purpose of this procedure is to induce human-like behaviors to the bot that uses the learned model. Additionally, we propose an exploration process—based on safe RL methods [14]—aimed to refine
the game model while maintaining induced human-like strategies. In regards to high-dimensional state–action spaces, HRLB^2 decomposes them into a set of smaller sub-problems using temporally extended actions [15]. Thus, the resulting hierarchical structure takes advantage of temporal abstraction and state abstraction.

The second challenge we found is generating varied behaviors, which also adapt to the opponent’s playing style. We approached this problem with the inclusion of a reward shaping mechanism [16,17]. We used this mechanism to define reward transformations that lead a bot behavior to approach the same problem in a particular way.

We evaluated the effectiveness of HRLB^2 in generating believable behaviors for NPCs in the domain of the 2D fighting game named Street Fighter IV. Accordingly, we implemented a bot in our framework and then assessed its human-likeness by performing a third-person Turing test. The results of the test demonstrate that our bot has a much more human-like behavior than the built-in AI agents. Furthermore, this conclusion led us to provide a first attempt to better explain how research on human-like behaviors may bring developments in reinforcement learning.

2. Related Work

In this paper, we present a framework, HRLB^2, with the aim of creating believable characters. In particular, we focus on player believability [18]; this characterization of believability implies the design of NPCs that display a human-like behavior, which also entails a believable bot may not be as intelligent as a human player. Nevertheless, in different contexts, it is more challenging to create human-like behaviors than highly skilled—even superhuman—NPCs [19].

The complexity of creating human-like behaviors for NPCs makes this challenge an interesting research problem. Besides, there is empirical evidence that indicates that players prefer to play with or against human-like NPCs [20]. Consequently, developing bots that appear to be controlled by a human player might benefit both advancements in AI research and the video game industry.

There have been many efforts to create believable bots in different game genres [19,21,22]. In a broad manner, these works are classified into direct and indirect behavior imitation [21]. The direct imitation approach consists of using supervised learning algorithms that take as input traces of human play. In contrast, the indirect imitation approach tackles this problem by maximizing a fitness function that evaluates the human-likeness of an NPC’s behavior. Our framework followed a direct imitation approach to build the transition and reward functions: we acquired data of human play traces to learn the system dynamics of a given game. On the other hand, the design of the needed reward functions involved an indirect imitation method: a reward function must capture the desired agent’s behavior, which RL uses as a fitness function.

A great example of current trends in human-like behaviors research is presented in [19], where the authors addressed the problem of creating believable bots; with the ability to play any game of the General Video Game-AI (GVG-AI) framework [23]. Particularly, in [19] a framework for human-like General Game AI is introduced that uses a modified version of the Monte Carlo Tree Search (MCTS) method. The proposed adjustments to MCTS consist on heuristics and quantitative measures of player behavior that bias the action selection to be more human-like.

The quantitative measures of player behavior that use the framework explained in [19] consist of analyzing human traces to compute the distributions different patterns of low-level actions. The authors of [19] found that the main low-level action patterns to consider are: action length, empty nil-action length and action to new action change frequency. Then, the computed distributions of these low-level actions are combined with MCTS to create believable and effective NPCs.

Likewise, research on human-like behaviors has been approached outside the Game AI community [24–26]. For instance, in [24] a method that creates human-like gaze behavior for a storytelling robot is presented. The objective of this method is to dictate how the robot should look at the members of the audience in a believable manner. To achieve this goal, the authors of [24]
proposed a direct imitation approach that combines data collected from a human storyteller and a discourse structure model.

HRLB\textsuperscript{2} approaches high-dimensional state–action spaces by decomposing them into a set of smaller sub-problems using temporally extended actions [27]. This procedure has been widely used to tackle large problems that can be represented at different levels of abstractions [28–30]. Furthermore, this hierarchical decomposition allows incorporating expert knowledge into the model and, in similar RL configurations to ours, it also reduces the exploration process without sacrificing learning performance [31,32]. Therefore, although it takes a lot of effort to incorporate expert’s knowledge in the form of a hierarchical decomposition of MDPs, this contributes to provide better solutions for complex problems that current automatic techniques would not be able to tackle [32].

Principally, the procedure of HRLB\textsuperscript{2} to induce human-like behaviors is similar to the work in [19], although the hierarchical structure of our framework allows inducing more abstract patterns of actions. However, this advantage comes with the difficulty of designing by hand the hierarchy, and reward functions, for previously unseen games. We believe this difficulty is acceptable since we are dealing with a more complex game than those in the GVG-AI framework [23].

Lastly, the closest work we have found to ours is [33]. The authors proposed three methods for believable agents that mix RL and supervised learning in different manners. The approach that achieved the best human-likeness score consists of a RL model and a neural network (NN) running in parallel. In the learning phase, the RL model learns to play from scratch by interacting with the environment, while the NN is trained with data from human behaviors. Throughout their planning, both algorithms sum up their output with the objective of biasing the RL model with the NN output.

3. Background and Notation

This section provides a brief description of the MDP model [34], the MAXQ approach on hierarchical reinforcement learning [27,29] and SPUDD [35].

3.1. MDP: Definition

An MDP is an optimization model for an agent acting in a stochastic environment and satisfying the Markov property. An MDP is defined by the tuple \( \langle S, A, T, R \rangle \), where:

- \( S \) is a set of states;
- \( A \) is a set of actions;
- \( T : S \times A \times S \to \{0, 1\} \) is the transition function that assigns the probability of reaching state \( s' \) when executing action \( a \) in state \( s \) that is, \( T(s' | s, a) = P(s' | a, s) \); and
- \( R : S \times A \to \mathbb{R} \) is the reward function, with \( R(s, a) \) denoting the immediate numeric reward value obtained when the agent performs action \( a \) in state \( s \).

A policy, \( \pi \), for an MDP is a function \( \pi : S \to A \) that specifies the corresponding action \( a \) to be performed at each state \( s \). Therefore, \( \pi(s) \) denotes the action \( a \) to be taken in state \( s \).

3.2. MAXQ Hierarchical Decomposition

The MAXQ hierarchical decomposition is a method for decomposing MDPs into a set of smaller semi-Markov Decision Processes (SMDPs) [27,29]. The SMDP is a generalization of MDPs with the inclusion of temporally extended actions; that is, actions may take more than one time step to complete. Specifically, the MAXQ method takes an MDP, \( M \), as its input and decomposes it into a finite set of subtasks \( \{M_0, M_1, ..., M_n\} \). These subtasks are represented as a SMDP taking \( M_0 \) as the root subtask. Therefore, solving the root subtask \( M_0 \) is equivalent to solving the original MDP \( M \). In particular, for this article, we use the MAXQ algorithm, and notation, presented in [29].

Since we are using a model-based approach, we need to be able to query \( R(s, a) \) and \( T(s' | s, a) \) to compute a model with both primitive and composite actions \( M_a \) so we can solve the graph of
hierarchical SMDPs. We can achieve this by computing $R(s,a)$ (Equation (4) in [31]) and $T(s' | s,a)$ (Equation (5) in [31]) for composite actions $M_a$ with:

$$R(s,a) = R(i, \pi_i^*(s)) + \sum_{s'} T(i, s' | s, \pi_i^*(s)) R(i, s')$$

(1)

$$T(s' | s,a) = P_t(s' | s,a)$$

(2)

where $\pi_i^*$ is the optimal policy for subtask $M_i$, $T(i, s' | s, \pi_i^*(s))$ is the transition function for subtask $M_i$ that assigns the probability of reaching a future state $s'$ when following $\pi_i^*$ from state $s$ and $P_t(s' | s,a)$ is called the *termination distribution* since it defines the marginal distribution over terminal states $G_i$ of subtask $M_i$. This distribution determines the probability that subtask $M_a$ will terminate at state $s'$ starting from state $s$.

For a more complete introduction to hierarchical reinforcement learning, please refer to [15].

3.3. SPUDD

Solving small MDPs with the classical methods is very efficient [36]; however, typical AI planning problems become intractable for this kind of implementations [35,36]. In [35] SPUDD, a value iteration implementation that solves MDPs using Algebraic Decision Diagrams (ADDs), is presented. This algorithm takes advantage of the compact manner to represent MDPs that ADDs offer.

ADDs are a generalization of binary decision diagram (BDD) [37] that can have terminal nodes with numeric values. A BDD is a data structure that encodes Boolean functions as rooted, directed, acyclic graphs. Furthermore, in SPUDD, all transition and reward functions are represented using ADDs, which are specified as *Lisp* trees using parenthesis.

For instance, the ADD displayed in Figure 1 should be defined in SPUDD as: $(w(a(-0.5))(b(-0.1))(c(0.0)))$. This ADD can be interpreted as a reward function, where the leafs provide the respective reward for each different value that variable $w$ can take.

![Figure 1. An algebraic decision diagram.](image)

3.4. Reward Shaping

Reward shaping is a method for guiding reinforcement learning to improve its learning rate and effectiveness of behaviors [16,17]. This form of advice is especially advantageous in highly stochastic environments [17], such as video games.

Although the design of hand-authored reward functions might be seen as providing RL with the solution to the problem at hand, there is empirical evidence that supports that advised reward functions will lead to similar policies to those found without advice [17]. That is, with enough time to learn, advised and unadvised agents will behave in a similar manner.

For this paper, we transform our reward functions as $R' = R + F$, where $F: S \times A \times S \to \mathbb{R}$ is a bounded real-valued function called the *shaping reward function* [16].
4. HRLB^2: A Reinforcement Learning Based Framework for Believable Bots

In this section, we describe our model-based framework called HRLB^2 (hierarchical reinforcement learning for believable bots). In particular, we explain how our framework is structured and how it should be used to solve problems that are defined as MDPs.

4.1. Overview

HRLB^2 approaches high-dimensional state–action spaces by decomposing them into a set of smaller sub-problems using temporally extended actions. Since our framework is based on the MAXQ hierarchical Decomposition [27], the original problem is represented as a task graph with subtasks or primitive actions as nodes. However, unlike MAXQ, HRLB^2 uses a model-based reinforcement learning (RL) method (analogous to [31]). Therefore, instead of directly learning a value function V(s) for each subtask, HRLB^2 first learns the transition function T(s′ | s, a) for a given hierarchy of an MDP \( M = \{M_0, M_1, ..., M_n\} \). Then, HRLB^2 solves these MDPs through SPUDD [35]: a value iteration algorithm that uses algebraic decision diagrams (ADDs) [38] to represent value functions and policies.

Mainly, we chose a model-based hierarchical decomposition because this approach let us represent subtasks in a human-readable data format. Specifically, SPUDD’s ADD representation of MDPs allowed us to include hand-coded Boolean functions specified as scheme trees, as we explained in Section 3.3. With this feature, we could direct the agent how to behave in particular situations. Hence, it became possible to correct unsuitable behaviors that arise from inaccuracies in system dynamics. Furthermore, even though the proposed model can be solved as a flat MDP, we preferred to adopt a hierarchical decomposition because the exploration process may become narrower without sacrificing learning performance [31]. In addition, this may increase the believability of behaviors since the exploration is constrained to the region explored by the observed human behaviors.

To compute the system dynamics of a given problem, we propose a two-step learning procedure. The first step is comprised of a data driven approach to estimate the transition function T(s′ | s, a). To do so, we acquire data, of human behaviors, by observing how humans play the game, for which we want to create a bot. With this model, we proceed to solve the problem at hand, that is, finding an optimal policy \( \pi^*_i \) for each subtask in the hierarchy. Nevertheless, the amount of data needed to get an accurate transition function is restrictive. Consequently, for the second learning step, the agent refines the transition function by exploring the environment.

Since exploring the environment in a random manner might lessen the human-like bias induced in the first step, we introduced a heuristic exploring function that incorporates advice from an expert. The expert’s advice is defined as a believable function that reduces the probability of performing actions that would lead the agent to an unknown state s in the environment. Furthermore, our exploring function encourages the exploration of rarely tried actions, in known states, by keeping visiting counts of state s and state–action pair (s, a) for each subtask. In this manner, the agent is able to experiment how well the knowledge acquired by observing the human demonstrator has been represented, while exploring new behaviors that remain believable.

Thus far, all the elements of HRLB^2 that we have explained are offline methods. Nevertheless, to create a bot that exhibits diversity of behaviors and adapts to its opponent’s playing style, our framework also includes an online update rule for the value function \( V^* (s) \) of the subtasks Mst_i. These special subtasks are designed to represent the performance of different playing styles that the agent can execute (see Figure 2). Consequently, we can adapt in real-time the playing style of the bot to achieve more effective and varied behaviors.
4.2. Hierarchical Decomposition

The first step to approach a problem through the HRL^2 method is constructing its corresponding task graph. This procedure allows us to tackle domains with high-dimensional state–action spaces and integrates expert’s knowledge about the environment, which may induce human-like strategies to the agent and reduce the exploration process. Furthermore, our hierarchical decomposition algorithm includes a layer that empowers the agent with multiple playing styles.

As we can see in Figure 2, the playing style layer includes the child nodes of the root MDP \( M_0 \). The set of nodes in the subtask layer below is \{ \( M_{st1}^{11}, M_{st1}^{12}, M_{st2}^{1}, M_{st2}^{2} \} \). The subscript in this naming notation represents the sub-problem it is intended to solve, while the superscript indicates the archetype behavior for the agent. Thus, the subtasks \{ \( M_{st1}^{11}, M_{st1}^{12} \} \) are designed to tackle the same state–action space, and achieve the same goal, but with different approaches.

Hence, the diversity of behaviors resides in the design of multiple subtasks with distinct directions to achieve the same goal for a particular sub-problem. To implement varied ways to approach the same sub-problem, we create appropriate reward functions that foster certain traits, which fit the corresponding playing style archetype we want the agent to exhibit. For instance, if we want to create a bot for Street Fighter IV with an aggressive fighting style, we should implement a reward function that encourages attacks at a close range over long range ones and defensive techniques.

4.3. Learning by Observation

As shown in Algorithm 1, a recursive procedure to learn the model of all subtasks \( M_i \) in the task graph is performed. This task is achieved by observing a human player while playing the game for which we want to create a believable bot. Therefore, its requires constantly observing the current state \( s \) of the game environment and detecting when the player begins the execution of an action \( M_a \) along with its corresponding reward \( R(s, M_a) \). Then, after the completion of action \( M_a \), we store in \( s' \) the new state that the character has reached.

With these data, \( \langle s, M_a, s', R(s, M_a) \rangle \), we proceed to update the model of the current subtask, as shown in Algorithm 2. Thus, the output of this algorithm is \( \langle T(s' | s, a), R(s, M_a) \rangle \). If we are dealing with primitive actions, this computation is straightforward. However, for composite actions, we need to calculate \( \langle T(s' | s, a), R(s, M_a) \rangle \) using Equations (2) and (1), respectively. Once the required observations have been obtained, the complete hierarchical model is exported in SPUDD format. In addition, it is worth mentioning that all observation data in \( \langle N_i[s, a, s'], N_i[s, a], R_i[s, a] \rangle \) are exported so we can continue later with the learning procedure.

Before we continue with the exploration process, we have to solve all subtasks \( M_i \) for all the different bot’s playing styles. That is, we need to find the policies \( \pi_i^*(s) \) that maximizes the expected reward for all states in the game’s domain. This is accomplished by the algebraic decision diagrams based value iteration algorithm of SPUDD [35].
Algorithm 1: Learn()

Input: a problem described in HRLB^2
Output: \( \langle T(s' | s, a), R(s, a) \rangle \)

\( s \leftarrow \text{ObserveState}() \)

if player initialize an action \( M_a \) then
  if \( M_a \) is primitive then
    \( s' \leftarrow \text{ObserveState}() \)
    \( v \leftarrow R(s, M_a) \)
    \( \langle t, v \rangle \leftarrow \text{UpdateM}(s, a, s', v) \)
    return \( \langle t, v \rangle \)
  else
    while \( s' \notin G_0 \) do
      \( \langle t, v \rangle \leftarrow \text{Learn}() \)
      \( \langle t, v \rangle \leftarrow \text{UpdateM}(s, a, s', v) \)
    return \( \langle t, v \rangle \)

Algorithm 2: UpdateM(s, a, s', v)

Input: \( \langle s, a, s', v \rangle \)
Output: computed \( \langle T(s' | s, a), R(s, a) \rangle \)

\( N_i[s, a, s'] \leftarrow N_i[s, a, s'] + 1 \)

\( N_i[s, a] \leftarrow N_i[s, a] + 1 \)

\( R_i[s, a] \leftarrow R_i[s, a] + v \)

if \( a \) is primitive then
  \( t \leftarrow \frac{N_i[s, a, s']}{N_i[s, a]} \)
  \( v \leftarrow \frac{R_i(s, a)}{N_i[s, a]} \)
  return \( \langle t, v \rangle \)

else
  \( t \leftarrow T(s' | s, a) \) using Equation (2)
  \( v \leftarrow R(s, a) \) using Equation (1)
  return \( \langle t, v \rangle \)

4.4. Heuristic Exploration Process

In this section, we explain in detail our proposed procedure that lets the agent explore the state–action space of a game without violating the human-like restrictions. These restrictions were induced in the transition model by following Algorithm 1. However, the necessary number of data to also achieve effective behaviors would be enormous for any modern video game, since, generally speaking, their state and action spaces are at least \( 10^{1685} \) \[32\]. Thus, it is crucial to achieve our main goal to refine the previously learned model by letting the agent explore the environment by itself without violating the human-like restrictions that limit the space of allowable policies to those that a human would perform.

Our proposed heuristic exploration process is based on a constrained criterion in which the expectation of return is maximized to one or more constraints \( c_i \in C \). According to García and Fernández \[14\], the generalization of this criterion is written as:

\[
\max_{\pi \in \Pi} E_\pi(R) \text{ subject to } c_i \in C, c_i = h_i \leq \alpha_i
\]  

(3)
where the set $C$ constrains all the constraint rules $c_i$ that the policy $\pi$ must fulfill with $c_i = h_i \leq \alpha_i$, with $h_i$ a function related with the return and $\alpha_i$ the threshold restricting the values of this function. Particularly, we follow a chance–constraint approach that allows with a certain probability breaking the constraint $c_i$. This method is shown in the following:

$$P(E(R) \geq a) \geq (1 - \epsilon)$$

That is, the expected return of the random variable $R$ will be at least as good as $a$ with a probability greater than or equal to $(1 - \epsilon)$ [39]. In our setting, this is interpreted as the action-value $Q(s, a)$, which is defined as the expected cumulative reward by performing action $a$ in state $s$ and then follow policy $\pi$ thereafter, being at least as good as $R(s, a)$ with a probability greater than or equal to $(1 - \epsilon)$, where $\epsilon = 0.15$, at the beginning of the process, and continually decreases until it reaches the value of 0. The reasoning behind the value of $\epsilon$ is that we only encourage to try actions $a$ that will not lead to states where the bot receives highly negative rewards. On the other hand, the variable value of $\epsilon$ is intended to facilitate acquiring new knowledge at the beginning of the exploration and refines the transition model $T(s' | s, a)$ by the end.

Furthermore, we add a biased bonus that favors the exploration of rarely used actions with the equation $\kappa \sqrt{\ln N_i[s] / N_i[s, a]}$. $N_i[s]$ and $N_i[s, a]$ are the visiting counts of state $s$ and state–action pair $(s, a)$, respectively. $\kappa$ is a constant value [0, 1] that determines magnitude of effect on the original Equation (4). Therefore, our chance–constraint equation is rewritten as:

$$P(E(R) \geq a) + \kappa \sqrt{\ln N_i[s] / N_i[s, a]} \geq (1 - \epsilon)$$

In Algorithms 3 and 4, we convey the complete process to our proposed heuristic exploration process for believable behaviors. We begin by observing the current state $s$ of the bot and then selecting, according to Equation (5), the—primitive or composite—action $M_a$ to perform. Once the bot finishes the execution of the action $M_a$, we observe again the state of the bot and store it in variable $s'$. In addition, we compute the corresponding reward $v$ for performing action $M_a$ and reaching state $s'$. Next, we update the transition and reward functions, $(T(s' | s, a), R(s, a))$, using the values in $(s, M_a, s', v)$.

| Algorithm 3: Explore() |
|-----------------------|
| **Input:** a problem described in HRLB^2 and policies $\pi_i$ |
| **Output:** $(T(s' | s, a), R(s, a))$ |
| $s \leftarrow$ ObserveState() |
| $a \leftarrow$ SelectAction($s, \epsilon$) |
| **if** $a$ **is primitive** **then** |
| ExecuteAction($a$) |
| $s' \leftarrow$ ObserveState() |
| $v \leftarrow R(s, a)$ |
| $(t, v) \leftarrow$ UpdateM($s, a, s', v$) |
| **return** $(t, v)$ |
| **else** |
| **while** $a$ **is executing do** |
| ExecuteAction(primitive($a$)) |
| $s' \leftarrow$ ObserveState() |
| $v \leftarrow R(s, a)$ |
| $(t, v) \leftarrow$ UpdateM($s, a, s', v$) |
| **return** $(t, v)$ |
Algorithm 4: SelectAction(s, ε)

Input: bot’s current state s
Output: a

\[ a \leftarrow \text{using Equation (5)} \]

return a

4.5. Online Planning Algorithm

The last component of HRL\textsuperscript{B2} is intended to choose the best action \( M_a \) and playing style \( M_{st_i} \) to maximize the expected bot’s reward. In Algorithm 5, the process to achieve this objective is shown. First, we observe the current state \( s \) and greedily select the best action \( M^*_a \) to perform according to the policy \( \pi^*_i(s) \). After performing action \( M^*_a \), we observe the environment again and compute the reward \( v = R(s, a) + R(s') \) that the bot acquired, where \( R(s, a) \) is the reward for performing action \( a \) in state \( s \) and \( R(s') \) represents the reward of reaching state \( s' \) after completing action \( a \).

Algorithm 5: Play()

Input: a problem described in HRL\textsuperscript{B2}, policies \( \pi^*_i \) and action-value functions \( Q_i(s, a) \)
Output: Updated \( Q_{st}(s, a) \)

\[ Q \leftarrow Q_{st}(s, a) \]
\[ Q_{st} \leftarrow 0 \]
\[ s \leftarrow \text{ObserveState()} \]
\[ a \leftarrow \pi^*_i(s) \]

if \( a \) is primitive then

- ExecuteAction(a)
- \( s' \leftarrow \text{ObserveState()} \)
- \( v \leftarrow R(s, a) + R(s') \)
- \( Q_{st} \leftarrow Q + \eta [v - Q] \)

return \( Q_{st} \)

else

while \( a \) is executing do

- ExecuteAction(primitive(a))
- \( s' \leftarrow \text{ObserveState()} \)
- \( v \leftarrow R(s, a) + R(s') \)
- \( Q_{st} \leftarrow Q + \eta [v - Q] \)

return \( Q_{st} \)

Then, with value \( v \), we proceed to update the action-value \( Q_i(s, a) \) of the corresponding playing style subtask \( M_{st_i}^{a_j} \). This update to the model is carried out by the next incremental learning rule

\[ Q_i(s, a) \leftarrow Q_i(s, a) + \eta [v - Q_i(s, a)] \]

where \( \eta \) is a constant value that represents the step-size of learning. This procedure empowers the bot with the ability to adapt to the playing style of its opponent. It is important to emphasize that this incremental learning rule only updates the action-value functions of the different playing styles.

Since human opponents adapt their strategies to try to overcome our bot game plan, we believe that the proposed global learning rule improves the diversity of behaviors of our bot—both players are changing their fighting approach in real time. Besides, we believe that the diversity of behaviors is key to improve the human-likeness of a bot.
5. Experiment: Street Fighter IV

In this section, we explain in detail the design and implementation of a believable bot based on the HRLB^2 architecture. Our bot was designed to play the fighting game Street Fighter IV, as shown in Figure 3. We assess its human-likeness with a third-person Turing.

Figure 3. A snapshot of Street Fighter IV.

Street Fighter IV as a Testbed for Believable Bots and Reinforcement Learning

The testbed we chose for our HRLB^2 architecture is the fighting game Street Fighter IV (SFIV). This game is a particularly difficult challenge since its state and action spaces are extensive. In addition, we should consider that it is a fast-paced game which leaves only 50–100 ms to make a decision. Moreover, SFIV is an imperfect and incomplete information game. Therefore, from a machine-learning standpoint, SFIV provides an excellent testbed for fast planning under uncertainty and cognitive skills based on the Theory of Mind [40].

Furthermore, creating believable bots for SFIV represents an especially challenging task. This difficulty arises from the fact that both players are always on screen in fighting games, making it harder to maintain the illusion of an agent being controlled by a human player. In contrast, in the FPS named Unreal Tournament—the testbed for the BotPrize Competition [41]—the judges and players participating in the Turing test only have a few moments to examine how their opponents play and react to the environment. Therefore, SFIV represents a more advanced challenge for the creation of believable bots.

6. Mark': A HRLB^2-Based agent for Street Fighter IV

The proposed case study for our architecture HRLB^2 is the design and implementation of a bot with the objective of playing SFIV in a human-like manner. We called our bot MarK; we chose this name after Markov and a fighting game character named K'.

Since SFIV is a highly complex domain, as a first step to approach the creation of believable characters through reinforcement learning, we focused on learning how to control, and play against, one particular character—Ryu. This solution reduced the action space to 70. Additionally, to reduce the state space, we used a coarser discretization method for the variables that track the position of the characters on screen. Despite these simplifications, we still faced a problem with a upper bound for the state space of $10^{1310}$. We computed SFIV state space complexity, using the number of possible states (512) and the number of cells (210), as $\log_{10}(512^{210})$—according to our discretization method that we present in Section 6.1. This simplified version of the state space of SFIV is still much larger than the state space of Go ($10^{170}$) or Chess ($10^{67}$).

To apply HRLB^2 to SFIV, we must first provide all the variables and subtasks that are needed to build the task hierarchy for the domain at hand.
6.1. Variables

In a fighting game such as SFIV, all the variables that represent the environment are associated with the features that define the state of both characters on screen. Since we considered the opponent as part of the environment, most of the variables that we present below have a variant for each of the characters:

- **Position**: The perception of the position of the NPCs is key to play a fighting game. In particular, MarK’ has to perceive the position of both agents in the vertical and horizontal axes.
- **Movement**: Similar to the last variables, it is important to incorporate knowledge about the direction of the NPCs’ movement in the horizontal and vertical axes.
- **Projectile**: This variable indicates if the opponent’s projectile is getting closer, getting farther or not moving, according to the other player perspective.
- **Bars**: There are three different bars in the game that quantify stats of the NPCs. The first measures the health level of the characters. Then, we have variables that gauge the amount of energy that can be used for special moves.
- **Frame Data**: Any of the actions in the game are not instantaneous and, during their execution time, they pass through three different phases: start-up, active and recovery. We discretized the phase of each move using these three variables.
- **Attacks**: In this category, we include all necessary variables to interpret the actions a NPC can execute. As our experiment limits to play with/against Ryu, we have 62 character specific actions.

6.2. Hierarchical Decomposition

In Figure 4, we present the task graph for MarK’. This hierarchical decomposition was proposed by our expert, who is also the first author of this paper. Therefore, this task graph incorporates domain knowledge that exploits state abstractions of individual MDPs within the hierarchy. With this procedure, we can significantly reduce the complexity of subtasks by ignoring parts of the state space that are not relevant to accomplish its goal [27]. Consequently, a well-designed hierarchical task is vital to achieve effective behaviors and tractable MDPs to solve.

Next, we present an overview of the designed subtasks to build the task graph for MarK’:

- **PrimitiveActions**: These ten actions are positioned in the lowest-level of the hierarchy. The eight-position joystick is used to control the movement of the NPC, while the rest of the buttons execute normals.
- **Normal(n), Cover(c), GoTo(x), JumpA(a, y), Special(s)**: Here, we have five different multi-step actions that we categorize as low-level subtasks. Normal(n) has the objective of performing the normal specified in the parameter n. Subtask Cover(t) is intended to block opponent’s attacks and takes parameter t as input, which represents the attack that the bot is about to receive so it can properly defend against it. GoTo(x) takes the bot to the specified position x. JumpA(a, y) has the goal of performing a jump attack; therefore, the parameters (a, y) represent the attack to perform and the elevation at which it has to be executed, respectively. Lastly, we have subtask Special(s), which executes the special attack s. Special attacks (specials) are more powerful than normals but also riskier and slower. All low-level subtasks only consider the state variables of the bot itself.
- **AntiAir, OppOTG, Stunt**: Here, we have three different multi-step actions that we define as high-level subtasks. These high-level subtasks have in common that all of them are implemented in only one playing style (Neutral). AntiAir specializes in defending against opponent’s jump attacks, that is attacks that are performed while the character is in a jump state. OppOTG is activated when the opponent is laying on the ground. Similarly, Stunt subtask activates when the opponent is in a stunt state. All high-level subtasks are designed to focus on specific situations that the bot may encounter; thus, they can ignore variables that are not relevant to accomplish their goal.
• **CloseRange, LongRange, BotOTG**: These high-level subtasks differ from the last ones because they are implemented in two different playing styles. As we can see in Figure 4, these multi-step actions have two variants: defensive and aggressive. *CloseRange* subtask focuses on viable strategies in a close range, while *LongRange* only considers effective long-range scenarios. *BotOTG* is activated when the bot is hit to the ground. Again, these kinds of subtasks ignore variables that are not relevant to accomplish their goal.

• **Defensive, Neutral, Aggressive**: Here, we have the subtasks that specify the playing style of their child actions. The *Defensive* playing style is intended to produce more cautious strategies for the bot. This results in an agent that is more passive and prefers to keep a longer distance between it and its opponent. Therefore, the agent spends more time performing subtask *LongRange*. On the other hand, the *Aggressive* playing style favors strategies that lead to giving damage to the opponent, regardless of how risky it may be. Thus, this version of the bot adopts more often attacks over defensive options from the subtask *CloseRange*. All the variables that use the child subtasks of each playing style are relevant.

• **Root**: This is the root task of the bot, that is the complete problem in a flat representation. Therefore, this MDP must consider all state variables.

With the design of the hierarchical decomposition for *MarK’*, we could proceed with the *learning by observation* procedure explained in Section 4.3.

![Figure 4](image_url). The task graph for our HRLB^2-based agent that plays *Street Fighter IV*.

### 6.3. Learning by Observation

The *learning by observation* process is aimed to learn the model of SFIV. In particular, this process was achieved by observing how our expert played SFIV against the built-in AI in the game at the available difficulty levels of 6–8. We selected this level of difficulty because, according to our expert, the built AI has exhibited the most human-like behavior in this configuration.

The learning process of the model was divided into 2-h intervals. After an interval is completed, all MDPs $M_i$ are solved through SPUDD. Then, for each interval, we evaluated the performance of the computed policies $\pi_i^*$ for all subtasks in the task graph. *MarK’* performance was estimated as the difference between the damage that it dealt and received in 10 rounds, fighting against the built-in AI level 6.
The first four epochs shown in Figure 5 are the learning rates of MarK’ over the 8-h period of learning by observation. We stopped the learning procedure after the fourth interval since the learning rate of our bot seemed to start slowing down. Next, we continued with the heuristic exploration process.

![Figure 5. Box plots of the learning rates of three different bots.](image)

6.4. Heuristic Exploration Process

The objective of following the heuristic exploration process, defined in Section 4.4, is to refine the previously computed model of SFIV by letting our bot to explore the environment by itself. In a similar fashion to the learning by observation process, the heuristic exploration process was divided into 2-h intervals. In addition, we computed the performance of our bot in the manner explained in Section 6.3.

As we can see in Figure 5, from Epoch 5 to Epoch 12, there were two box plots per epoch. The box plots on the right display the performance of MarK’ in each epoch, while the box plots on the left represent the same measure for a bot that uses a random exploration process.

For both bots, the total time spent in exploration was 16 h, or eight epochs. We stopped the exploration process at this point because the performance of MarK’ made a substantial improvement from the seventh to the eighth epoch.

In the next subsection, we explain the composition of the reward functions that aim to foster specific behavior traits to generate diverse playing styles.

6.5. Reward Functions

A well-designed reward function is key to achieve the desired behavior for our bot. Although there is no accepted definition of a proper design of reward functions, it is better when a reward function is kept straightforward. If a reward function remains simple enough, we can potentially use it for different problems, which is favorable for benchmarks. Consequently, our reward function only considers three universal elements in 2D fighting games that are essential to evaluate the performance of a character:
• Health: Health bars are virtually a must in fighting games since they indicate the remaining stamina for each character on screen. When a character runs out of stamina, it loses. The variables we use to represent the rewards of this element are: dealt damage by MarK’ ($R_{D+}(s')$) and taken damage by MarK’ ($R_{D-}(s')$).

• Cover: Even though dealing damage is crucial to win, covering is an effective technique for reducing the amount of received damage. The variables we use to represent the rewards of this element are: attack covered by MarK’ ($R_{C+}(s')$) and attack covered by opponent ($R_{C-}(s')$).

• Positioning: Keeping the right distance to your opponent is a fundamental ability to take advantage of the specific set of skills of each character.

To cope with the requirements described above, the design of our reward functions is based on a combination of shaped and sparse rewards. Sparse rewards are appropriate to represent health and cover elements. Table 1 displays the neutral reward function $R(s')$ and the corresponding shaping functions $F(s')$ to create the fighting style described in Section 4.2.

| Fighting Style | Rewards |
|----------------|---------|
| Neutral        |         |
| $R_{D+}(s')$   | 1.0     |
| $R_{D-}(s')$   | $-1.17$ |
| $R_{C+}(s')$   | 1.0     |
| $R_{C-}(s')$   | $1.5$   |
| Defensive      |         |
| $F_{D+}(s')$   | 0.0     |
| $F_{D-}(s')$   | $-0.5$  |
| $F_{C+}(s')$   | $-0.77$ |
| $F_{C-}(s')$   | $-0.5$  |
| Aggressive     |         |
| $F_{D+}(s')$   | 0.0     |
| $F_{D-}(s')$   | 0.0     |
| $F_{C+}(s')$   | 0.0     |
| $F_{C-}(s')$   | $-0.5$  |

On the other hand, shaped rewards are necessary to describe a good positioning—we give increasing reward $r$ in ranges that are closer to the optimal position for fighting. We use an exponential function to define shaped rewards. This exponential is defined so the bot only gets a bias $F(s') = 0.01$ when the difference between the optimal and current positions and the current is maximum and it gets a bias $F(s') = 0.15$ when reaching the optimal position. When the Close Range macro-action is active, the optimal position for the bot is 6 units. For the Long-Range macro-action, the optimal position is 13 units.

6.6. Boolean Functions

Below, we explain the only Boolean function we designed for MarK’:

• AntiAirFunction: This function is intended to assist MarK’ on being more consistent in anti-air techniques. Since we noticed that our bot was bad at defense when its opponent jumped towards it from a long distance, we coded an ADD that compensated the use of the action called Shoryuken when the opponent was close enough to be hit by this attack.

6.7. Implementation

In regard to the hand-authoring design of macro-actions, reward functions and Boolean functions, we spent a total of 170 h to build everything. Although this amount of work can be considered as excessive, we should consider that most of this time was used to solve the MDPs of the task graph multiple times; it takes about 20 h to solve the corresponding MDPs for all playing styles of MarK’. This iterative process to find the best MDPs that better model a human-like behavior, with different playing styles, had to be repeated around 10 times. That is, nearly 160 h were spent solving the MDPs through SPUDD.
7. Assessing Believability

In this section, we explain the third-person Turing test we performed to assess our HRLB^2-based bot’s level of human-likeness. We used this information as a baseline to compare against the believability ratios of three different difficulty levels of the built-in AI of SFIV and three different human players with diverse skill levels. In addition, it is worth mentioning that 171 people participated as judges in our third-person Turing test; and the main findings of this study are presented in Figure 6 and Table 2.

![Figure 6](image)

**Figure 6.** Computed human-likeness ratios for the third-person Turing test.

7.1. Third-Person Turing Test

This variation of the Turing test for bots is considered as a third-person configuration because judges do not play against the subjects to be evaluated; they only observe how they play. The seven participants in this believability experiment are: MarK', a beginner human player (Human 1), an intermediate level player (Human 2), an advanced player (Human 3), the built-in AI of SFIV (CPU bot) set to level 2 (CPU 2), CPU bot level 4 (CPU 4) and CPU bot level 6 (CPU 6).

Our selection of participants was intended to provide a wide sample from the distinct behaviors that humans and bots can exhibit depending on their skill level. As a result, we could compare the human-likeness level of MarK’ to provide consistent behavior baselines that, we believe, can be adapted as standards.

A fundamental element of our third-person Turing test was video recordings of matches where each participant fights all other participants. Hence, our survey included 21 match combinations and, for each of them, we recorded two different fights to acquire an extensive sample of behaviors from all players.

In addition, it is important for a Turing test to know who is taking the test. Hence, we published our survey on specialized channels of the Fighting Game Community (FGC). In this manner, we
consider that most of the persons taking the Turing test have previous expertise on fighting games in general.

Our survey was published online and consists on showing, in a random manner, a match from our 42 unique videos. After the completion of the match, we present the user the next obligatory fixed-choice questions in relation to the video they just watched:

- How would you assess the fighting skill level of Player 1? with choices: Beginner, Intermediate, Advanced and Professional
- How would you assess the fighting skill level of Player 2? with choices: Beginner, Intermediate, Advanced and Professional
- Which character do you consider most likely to have been controlled by a human player? with choices: Player 1, Player 2

Furthermore, we included the next optional open-ended question:

- If you could kindly provide us with a deeper insight about the reasons that made you choose a player as more likely to have been controlled by a human, please leave a comment below

After these questions, we repeated this process two more times. That is, each user assessed the human-likeness of players in three different matches chosen at random from our set of videos. Then, we concluded the survey with the following obligatory fixed-choice question:

- How would you assess your skills as a fighting game player? with choices: Beginner, Intermediate, Advanced, Professional

Our survey was completed 171 times which implies we collected 513 match assessment evaluations. Moreover, the optional open-ended question was answered by 78 people. With these data, we proceeded to evaluate the human-likeness level of MarK’, the CPU bot of SFIV and human players.

### 7.2. Results of the Third-Person Turing Test

The first measure that we computed for comparison was the human-likeness ratios for the seven participants in the believability test. This ratio is estimated as $h/n$, where $h$ represents the number of times a participant was considered human, and $n$ the total number of times a participant appeared in an evaluated match. Figure 6 shows the results of the computed human-likeness ratios; as we can see, the CPU bots were considered much less human than MarK’ and the human players. In fact, MarK’ got a higher score than Human 1 and Human 2. However, the most skilled human player, Human 3, got the highest human-likeness ratio (0.67153).

Although MarK’ achieved a higher believability ratio than all the CPU bots, we needed to perform a test to validate its statistical significance. Given the features of our data, we chose to apply the two-tailed Fisher’s exact test, for the analysis of the significance related to the association between the human-likeness scores of all different participants in the third-person Turing test. In Table 2, we report the estimated $p$-values for MarK’ against the rest of participants. We rejected the null hypothesis with a 5% of significance level.

| Comparison      | $p$-Value | Cohen’s d | 95% CI  | Cohen’s d | Size of Effect |
|-----------------|-----------|-----------|---------|-----------|----------------|
| MarK’ vs. Human 1 | 0.03857   | -2.031-0.629 | -0.746 | Medium    |
| MarK’ vs. Human 2 | 0.02205   | -1.563-1.253 | -0.065 | Small     |
| MarK’ vs. Human 3 | 0.5358    | -1.071-1.372 | 0.301  | Small     |
| MarK’ vs. CPU 2  | $< 0.05$  | -3.937-1.022 | -2.476 | Large     |
| MarK’ vs. CPU 4  | $< 0.05$  | -3.285-1.318 | -2.323 | Large     |
| MarK’ vs. CPU 6  | $< 0.05$  | -2.606-0.268 | -2.180 | Large     |

1 All $p$-values less than 0.05 are in bold. 2 All 95% Confidence Intervals that exclude 0 are in bold.
According to the computed \( p \)-values, MarK’ plays with a style closer to Human 3 and far distant to CPU bots. Considering that MarK’ learned how to play SFIV by observing Human 3, we can argue that the exploration method of HRLB^2 is effective in maintaining the bias derived from human observation of samples.

Although reporting \( p \)-values is the standard procedure for this kind of research in the game AI community, we have also included the analysis of effect size and confidence intervals (CIs). This approach let us quantify the difference between the behaviors of our participants in the third-person Turing test because the effect’s size favors the difference of sizes instead of confounding this with sample size [42]. Specifically, we applied the bootstrap effect sizes algorithm (bootES) implementation of [43] to our data. Additionally, we standardized this effect’s size for the Cohen’s \( d \) measure.

In Table 2, we present the estimated Cohen’s \( d \), 95% CIs and the size effect for MarK’ against the rest of participants. From these measures, we conclude that the manner to play SFIV of MarK’ is highly dissimilar to the CPU bots’ playing style. On the other hand, our bot’s behavior is comparable to the playing style of the three human participants. The size effect is minimal when compared to Human 2 and Human 3. In other words, MarK’ plays in an analogous style to intermediate and advanced players.

As we have stated before, our main goal in this research work was to create a believable agent that plays in an effective manner. In addition, we built violin plots to better evaluate the exhibit skill level of our HRLB^2-based bot.

In Figure 7, we present a violin plot that presents the participants’ skill level and the self-evaluation skill level of the judges. In this plot, the median is shown as a white dot, the thick bar in the center represents the interquartile range and the thin line represents the 95% confidence interval. By visually inspecting our violin plot, we can notice that the density distribution for MarK’ is in between Human 2 and Human 3. Because of this analysis—and the previous size effect study—we can assert that MarK’ exhibited a higher skill level than Human 2 but not as good as Human 3. Although MarK’ could not match its teacher’s competence (Human 3), we believe that the attained skill level is good enough to be considered as effective. However, there is still room for improvement in this regard.

![Violin plot](image-url)

**Figure 7.** Third-person Turing test violin plot.

Lastly, we have read the 117 comments, by 78 people, from the open-ended question. From the analysis of these comments, we formulated the main reasons why our HRLB^2-based bot was
considered as a human player or a computer. Additionally, we include the number of times each comment was observed. These conclusions are listed below:

MarK’ was classified as a human mainly because it:

- Followed strategies and performed combos that are suitable for a person of its exhibited skill level (15 observations)
- Exhibited a wider range of attacks than its opponent (11 observations)
- Made execution mistakes that a human would make (8 observations)
- Looked like it adapted to its opponent’s game style (7 observations)

MarK’ was classified as a computer mainly because:

- Its anti-air strategies were very consistent (11 observations)
- Made errors that a person of its exhibited skill level would not make (9 observations)
- It seemed like it did not predict its opponents moves (4 observations)
- It did not expressed emotions, such as fear when it was about to lose (3 observations)

8. Evaluation of HRLB^2

In this section, we present the implemented experiment that are aimed to better understand the individual contribution of HRLB^2’s modules to the human-likeness and performance of MarK’. For this experiment, we could collect 97 responses of people that participated as judges. Furthermore, the main results of this study are presented in Figures 8 and 9 and Table 4.

![Believability Ratio Chart](chart.png)

**Figure 8.** Computed human-likeness ratios for the third-person Turing test for HRLB^2.
Figure 9. Evaluation of HRLB^2 violin plot.

8.1. Third-Person Turing Test for HRLB^2

We chose to perform a third-person Turing test for bots to evaluate the modules of our architecture. The participants in this believability analysis are the next bots:

- MarK': This is our bot that is designed to employ all the modules of HRLB^2.
- Bot 1: This bot employs all the modules of HRLB^2; however, the exploration process is implemented in a random manner.
- Bot 2: This bot is exactly the same as MarK', but without the online planning algorithm that adapts the playing style of the bot in real-time.
- Bot 3: This bot is exactly the same as MarK', but without the Boolean function that improves the use of anti-air techniques.
- Bot 4: This bot is implemented with a random exploration process, without the playing style adaptation module and without the Boolean function.
- Bot 5: This bot is implemented with our proposed heuristic exploration process, without the playing style adaptation module and without the Boolean function.

We recorded two different videos per each participant in this Turing test. In these videos, all participants played against the built-in AI level 6 (CPU 6). Therefore, this Turing test included 12 unique samples of bot behaviors.

We published our survey on specialized channels of the FGC, to ensure the respondents have a strong background in fighting games. Before evaluating the behavior of the participants, we explained to the respondents that in all videos a bot was playing against the CPU 6. Thereafter, we presented, in a random manner, two matches from our set of videos. After the completion of the two matches, we asked the respondent the obligatory fixed-choice questions:

- Which bot played in a more human-like manner against the CPU? with choices: Bot in first video, and Bot in second video
- How would you assess the fighting skill level of the bot in the first video? with choices: Beginner, Intermediate, Advanced and Professional
• How would you assess the fighting skill level of the bot in the second video? with choices: Beginner, Intermediate, Advanced and Professional

In addition, we included the next optional open-ended question:

• If you could kindly provide more insight about the reasons that made you choose a bot as more human-like, please leave a comment below

Finally, this Turing test finished with the following obligatory fixed-choice question:

• How would you assess your skills as a fighting game player? with choices: Beginner, Intermediate, Advanced and Professional

We could collect 97 responses and the optional open-ended question was answered by 53 people. With these data, we proceeded to analyze the differences in human-likeness, and skill level, of all the implemented bots.

8.2. Results of the Third-Person Turing Test for HRLB^2

In the same manner as presented in Section 7.2, we computed the human-likeness ratios for the six bots in the believability test; these ratios are presented in Figure 8. Furthermore, we applied the same statistical significance analyses presented in Section 7.2. We report the estimated p-values and CIs in Table 3. In regards to performance, we present in Figure 9 a violin plot for each participant in this survey. In addition, this plot includes the self-assessment skill level of the judges.

Table 3. Human-likeness Analysis for HRLB^2.

| Comparison      | p-Value | Cohen's d 95% CI | Cohen's d | Size of Effect |
|-----------------|---------|------------------|-----------|----------------|
| Mark' vs. Bot 1 | 1       | -1.648–1.323     | -0.252    | Small          |
| Mark' vs. Bot 2 | 0.23535 | -2.632–0.38      | -1.054    | Large          |
| Mark' vs. Bot 3 | 0.844165| -1.191–1.847     | -0.375    | Small          |
| Mark' vs. Bot 4 | 0.48336 | -2.482–0.375     | -0.96     | Large          |
| Mark' vs. Bot 5 | 0.34599 | -2.692–0.205     | -1.222    | Large          |
| Bot 4 vs. Bot 5 | 1       | -1.812–1.621     | -0.008    | Small          |

1 All p-values less than 0.05 are in bold. 2 All 95% Confidence Intervals that exclude 0 are in bold.

Although Mark' and Bot 3 got the highest human-likeness ratios, the performed significance analyses do not find a significant difference against the rest of bots in the survey for HRLB^2. Nevertheless, the effect size is large when comparing Mark' against Bot 2, Bot 4 and Bot 5. Therefore, this might imply that the online planning algorithm—described in Section 4.5—is the module that contributes the most to the believability of our HRLB^2 bot—Mark'.

It is important to notice that, even if the human-likeness ratios are similar among the participants, the perceived skill level varies between the bots. By visually inspecting the violin plots in Figure 9, there is a noticeable difference between the density distributions of the bots. In addition, to better understand the magnitude of the variation in the skill level between the bots, we performed an effect size analysis and confidence intervals. The results of this analysis are presented in Table 4. A major feature to notice is that only Bot 3 has a comparable skill level to Mark'. This finding suggests that Mark' does not significantly improve its human-likeness or performance by using the proposed ADD in Section 6.6.

In addition, we can notice that there is a significant medium size effect between Mark' and Bot 1, as well as Bot 4 and Bot 5. Hence, we can affirm that our heuristic exploration method—explained in Section 4.4—achieves a better performance than its random counterpart. With this claim, we mean that the bots that use our heuristic exploration method exhibit a higher skill level that those that use a random exploration process. Since the raw performance of Bot 4 and Bot 5 are similar (see
Figure 5), we consider that our heuristic exploration method is better at biasing the RL model with the gathered human behavior observation. Based on this result, we believe that our learning by observation procedure, proposed in Section 4.3, is effective.

With a further analysis of the skill level data, we can affirm that our online planning algorithm (Section 4.5) is key to improving the perceived performance of a bot, since the size of effect is large when comparing MarK’ against Bot 2.

Table 4. Skill Level Analysis for HRLB^2.

| Comparison         | Cohen’s d | 95% CI   | Cohen’s d | Size of Effect |
|--------------------|-----------|----------|-----------|----------------|
| MarK’ vs. Bot 1    | −1.215    | −0.256   | −0.718    | Medium         |
| MarK’ vs. Bot 2    | −1.444    | −0.381   | −0.890    | Large          |
| MarK’ vs. Bot 3    | −0.905−0.051 | −0.447    | Small     |
| MarK’ vs. Bot 4    | −2.339−1.046 | −1.674    | Large     |
| MarK’ vs. Bot 5    | −1.729−0.651 | −1.180    | Large     |
| Bot 4 vs. Bot 5    | −1.043−0.067 | −0.547    | Medium    |

*All 95% Confidence Intervals that exclude 0 are in bold.*

9. Conclusions and Future Work

We achieved several positive outcomes with this work. Our most significant accomplishment was MarK’: a HRLB^2-based bot that plays Street Fighter IV in a human-like manner, with a medium-to-advanced fighting performance. Furthermore, the results of the analyses in Section 8 validate all the proposed modules of our architecture—each module contributes to the creation of a believable bot with a medium-to-advanced skill level.

In addition, our findings have arisen promising future directions. For instance, we would like to adapt HRLB^2 to be practical for human-like General Game AI. As a first-step to achieve this, it would be advantageous to reduce the amount of human intervention in the creation of macro-actions and reward functions. For example, there have been positive results in the automation of finding macro-actions [44], in combination with a function approximation [45] to be able to solve RL problems with state–action spaces of any size. Likewise, using an inverse reinforcement learning [46] approach would let us learn the reward function by observing human play traces of a given game. Besides, we could use the Video Game Description Language (VGDL) [47] to facilitate the use of HRLB^2 in different games.

Ultimately, we would like to conduct further analysis to validate how well the knowledge of MarK’ can transfer to other characters in SFIV. In addition, we would like to investigate if our hierarchical architecture is suitable for learning skills. As a first approximation, we would design higher level macro-actions to model fighting skills that are key to play—at an advanced level—most characters in SFIV.

Author Contributions: C.A.C. conducted this research work and wrote the paper under the supervision of J.A.R.U.

Funding: This research was funded by Tecnologico de Monterrey, Mexico. Also, the authors would like to thank the Consejo Nacional de Ciencia y Tecnologia (CONACYT) and, the Consejo Mexiquense de Ciencia y Tecnologia (COMECYT) for their support through the financial aids they provided.

Acknowledgments: The authors would like to acknowledge the support of the Computer Science Department, Tecnologico de Monterrey, Campus Estado de Mexico, Carr. Lago de Guadalupe km 3.5, Atizapan de Zaragoza 52926, Mexico, in the production of this work.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Cruz, C.A.; Uresti, J.A.R. Player-centered game AI from a flow perspective: Towards a better understanding of past trends and future directions. *Entertain. Comput.* 2017, 20, 11–24. [CrossRef]
2. Csikszentmihalyi, M. *Flow: The Psychology of Optimal Experience*; Harper Perennial: New York, NY, USA, 1990.
3. Conroy, D.; Wyeth, P.; Johnson, D. Modeling Player-like Behavior for Game AI Design. In Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology, Lisbon, Portugal, 8–11 November 2011. [CrossRef]

4. Laird, J.E.; Duchi, J.C. Creating Human-Like Synthetic Characters With Multiple Skill-Levels: A Case Study Using the Soar Quakebot; AAAI: Ann Arbor, MI, USA, 2000.

5. Schrum, J.; Karpov, I.V.; Miikkulainen, R. UT2: Human-like Behavior via Neuroevolution of Combat Behavior and Replay of Human Traces. In Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG 2011), Seoul, Korea, 31 August–3 September 2011; pp. 329–336.

6. Schrum, J.; Karpov, I.; Miikkulainen, R. Human-Like Combat Behaviour via Multiobjective Neuroevolution. In Believable Bots; Hingston, P., Ed.; Springer: Berlin/Heidelberg, Germany, 2012; pp. 119–150. [CrossRef]

7. Berseth, G.; Haworth, M.B.; Kapadia, M.; Faloutsos, P. Characterizing and Optimizing Game Level Difficulty. In Proceedings of the Seventh International Conference on Motion in Games, Playa Vista, CA, USA, 6–8 November 2014; pp. 153–160. [CrossRef]

8. Diaz-Furlong, H.; Solis-Gonzalez Cosio, A. An approach to level design using procedural content generation and difficulty curves. In Proceedings of the 2013 IEEE Conference on Computational Intelligence in Games (CIG), Niagara Falls, ON, Canada, 11–13 August 2013; pp. 1–8. [CrossRef]

9. Yang, J.; Gao, Y.; He, S.; Liu, X.; Fu, Y.; Chen, Y.; Ji, D. To Create Intelligent Adaptive Game Opponent by Using Monte-Carlo for Tree Search. In Proceedings of the ICNC ’09 Fifth International Conference on Natural Computation, Tianjin, China, 14–16 August 2009; Volume 5, pp. 603–607.

10. Liu, X.; Li, Y.; He, S.; Fu, Y.; Yang, J.; Ji, D.; Chen, Y. To Create Intelligent Adaptive Game Opponent by Using Monte-Carlo for the Game of Pac-Man. In Proceedings of the ICNC ’09 Fifth International Conference on Natural Computation, Tianjin, China, 14–16 August 2009; Volume 5, pp. 598–602.

11. Thrun, S. Learning to Play the Game of Chess. In Proceedings of the 7th International Conference on Neural Information Processing Systems; MIT Press: Cambridge, MA, USA, 1994; pp. 1069–1076.

12. McPartland, M.; Gallagher, M. Reinforcement Learning in First Person Shooter Games. IEEE Trans. Comput. Intell. AI Games 2011, 3, 43–56. [CrossRef]

13. Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. Mastering the game of go without human knowledge. Nature 2017, 550, 354–359. [CrossRef] [PubMed]

14. García, J.; Fernández, F. A comprehensive survey on safe reinforcement learning. J. Mach. Learn. Res. 2015, 16, 1437–1480.

15. Sutton, R.S.; Precup, D.; Singh, S. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. Artif. Intell. 1999, 112, 181–211. [CrossRef]

16. Ng, A.Y.; Harada, D.; Russell, S. Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping; ICML: Long Beach, CA, USA, 1999; Volume 99, pp. 278–287.

17. Wiewiora, E.; Cottrell, G.W.; Elkan, C. Principled methods for advising reinforcement learning agents. In Proceedings of the 20th International Conference on Machine Learning (ICML-03), Washington, DC, USA, 21–24 August 2003; pp. 792–799.

18. Togelius, J.; Yannakakis, G.; Karakovskiy, S.; Shaker, N. Assessing Believability. In Believable Bots; Hingston, P., Ed.; Springer: Berlin/Heidelberg, Germany, 2012; pp. 215–230. [CrossRef]

19. Khalifa, A.; Isaksen, A.; Togelius, J.; Nealen, A. Modifying MCTS for Human-Like General Video Game Playing. In Proceedings of the IJCAI, New York, NY, USA, 9–15 July 2016; pp. 2514–2520.

20. Arrabales, R.; Muñoz, J.; Ledezma, A.; Gutierrez, G.; Sanchis, A. A Machine Consciousness Approach to the Design of Human-Like Bots. In Believable Bots; Hingston, P., Ed.; Springer: Berlin/Heidelberg, Germany, 2012; pp. 171–191. [CrossRef]

21. Togelius, J.; De Nardi, R.; Lucas, S.M. Towards automatic personalised content creation for racing games. In Proceedings of the IEEE Symposium on Computational Intelligence and Games, Honolulu, HI, USA, 1–5 April 2007; pp. 252–259.

22. Ortega, J.; Shaker, N.; Togelius, J.; Yannakakis, G.N. Imitating human playing styles in super mario bros. Entertain. Comput. 2013, 4, 93–104. [CrossRef]

23. Perez-Liebana, D.; Samothrakis, S.; Togelius, J.; Schaul, T.; Lucas, S.M.; Couëtoux, A.; Lee, J.; Lim, C.U.; Thompson, T. The 2014 general video game playing competition. IEEE Trans. Comput. Intell. AI Games 2016, 8, 229–243. [CrossRef]
24. Mutlu, B.; Forlizzi, J.; Hodgins, J. A storytelling robot: Modeling and evaluation of human-like gaze behavior. In Proceedings of the 2006 6th IEEE-RAS International Conference on Humanoid Robots, Citeseer, Genova, Italy, 4–6 December 2006; pp. 518–523.

25. Potkonjak, V.; Tzafestas, S.; Kostic, D.; Djordjevic, G. Human-like behavior of robot arms: General considerations and the handwriting task—Part I: Mathematical description of human-like motion: Distributed positioning and virtual fatigue. Robot. Comput.-Integr. Manuf. 2001, 17, 305–315. [CrossRef]

26. Li, T.H.; Chang, S.J.; Chen, Y.X. Implementation of human-like driving skills by autonomous fuzzy behavior control on an FPGA-based car-like mobile robot. IEEE Trans. Ind. Electr. 2003, 50, 867–880. [CrossRef]

27. Dietterich, T.G. Hierarchical reinforcement learning with the MAXQ value function decomposition. J. Artif. Intell. Res. 2000, 13, 227–303. [CrossRef]

28. Mousas, C.; Anagnostopoulos, C.N. Real-time performance-driven finger motion synthesis. Comput. Graphics 2017, 65, 1–11. [CrossRef]

29. Bai, A.; Wu, F.; Chen, X. Online planning for large markov decision processes with hierarchical decomposition. ACM Trans. Intell. Syst. Technol. (TIST) 2015, 6, 45. [CrossRef]

30. Lee, Y.S.; Cho, S.B. Activity recognition using hierarchical hidden markov models on a smartphone with 3D accelerometer. In International Conference on Hybrid Artificial Intelligence Systems; Springer: Berlin/Heidelberg, Germany, 2011; pp. 460–467.

31. Jong, N.K.; Stone, P. Hierarchical model-based reinforcement learning: R-max+ MAXQ. In Proceedings of the 25th international conference on Machine learning, Helsinki, Finland, 5–9 July 2008; pp. 432–439.

32. Usunier, N.; Synnaeve, G.; Lin, Z.; Chintala, S. Episodic Exploration for Deep Deterministic Policies for StarCraft Micromanagement. arXiv 2016, arXiv:1609.02993.

33. Miyashita, S.; Lian, X.; Zeng, X.; Matsubara, T.; Uehara, K. Developing game AI agent behaving like human by mixing reinforcement learning and supervised learning. In Proceedings of the 2017 18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), Kanazawa, Japan, 26–28 June 2017; pp. 489–494. [CrossRef]

34. Puterman, M.L. Markov Decision Processes: Discrete Stochastic Dynamic Programming; John Wiley & Sons: Hoboken, NJ, USA, 2014.

35. Hoey, J.; St-Aubin, R.; Hu, A.; Boutilier, C. SPUDD: Stochastic planning using decision diagrams. arXiv 1999, arXiv:1301.6704.

36. Sucar, L.E. Probabilistic Graphical Models. In Advances in Computer Vision and Pattern Recognition; Springer: London, UK, 2015.

37. Bryant, R.E. Graph-based algorithms for boolean function manipulation. IEEE Trans. Comput. 1986, 100, 677–691. [CrossRef]

38. Bahar, R.I.; Frohm, E.A.; Gaona, C.M.; Hachtel, G.D.; Macii, E.; Pardo, A.; Somenzi, F. Algebric decision diagrams and their applications. Formal Methods Syst. Des. 1997, 10, 171–206. [CrossRef]

39. Delage, E.; Mannor, S. Percentile optimization for Markov decision processes with parameter uncertainty. Oper. Res. 2010, 58, 203–213. [CrossRef]

40. Leslie, A.M. Pretending and believing: Issues in the theory of ToMM. Cognition 1994, 50, 211–238. [CrossRef]

41. Hingston, P. The 2K BotPrize. In Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games, Milan, Italy, 7–10 September 2009. [CrossRef]

42. Coe, R. It’s the effect size, stupid: What effect size is and why it is important. Presented at the Annual Conference of the British Educational Research Association, University of Exeter, Exeter, UK, 12–14 September 2002

43. Kirby, K.N.; Gerlanc, D. BootES: An R package for bootstrap confidence intervals on effect sizes. Behav. Res. Methods 2013, 45, 905–927. [CrossRef] [PubMed]

44. Vezhnevets, A.; Mnih, V.; Osindero, S.; Graves, A.; Vinyals, O.; Agapiou, J. Strategic attentive writer for learning macro-actions. arXiv 2016, arXiv:1606.04695.

45. Sutton, R.S.; McAllester, D.A.; Singh, S.P.; Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In Proceedings of the NIPS’99 Proceedings of the 12th International Conference on Neural Information Processing Systems, Denver, CO, USA, 29 November–4 December 1999; pp. 1057–1063.
46. Abbeel, P.; Ng, A.Y. Inverse reinforcement learning. In Encyclopedia of Machine Learning; Springer: Berlin/Heidelberg, Germany, 2011; pp. 554–558.

47. Schaul, T. A video game description language for model-based or interactive learning. In Proceedings of the 2013 IEEE Conference on Computational Intelligence in Games (CIG), Niagara Falls, ON, Canada, 11–13 August 2013; pp. 1–8.

© 2018 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).