The Huddle: Combining AI Techniques to Coordinate a Player's Game Characters

Timothy Davison, Jörg Denzinger

***Abstract*—We present the *huddle*, a concept for extending games in which the player is responsible for a group of game characters. The huddle combines several AI methods to allow the player to create a cooperative strategy for his characters to solve a scenario of the game and it takes away from the player the need to frantically jump around in controlling his characters to employ the strategy idea he has. The huddle is entered from a saved game state and allows the player to provide his characters with strategy ideas in form of situations and the actions he wants the characters to take (SAPs). A learner then uses these ideas and adds to it additional SAPs to create a complete strategy. The learner uses a simulation of the real game that uses models for the non-player characters based on the experiences the player had with the game, so far, to evaluate strategy candidates. We evaluated the huddle idea with a fantasy-themed role playing game and show that the huddle indeed allows a player to concentrate on his strategy while still requiring him to come up with the solution ideas for scenarios.**

I. Introduction

Requiring the human player to control several game characters "simultaneously" is a feature of many of today's games. Many sports games need to deal with teams, simply because this is how the particular sport is played, but also other types of games use this feature, since it offers selecting the right team as an additional challenge and allows for losing team members (in addition to reflecting how the particular situations would be tackled in real life). But, for a single human player, controlling all his characters can become very difficult and the concepts used by game designers to support the player in this are far from perfect. Either the game offers pre-defined formations and character behaviors so that the task of the player is simply to find the right combination of these or the necessary cooperative behavior for solving a game scenario or situation is designed to be achievable by having the player switch direct control between his characters fast enough to enact it. While the latter can be very frustrating for some players, the former is often either too easy or too unrealistic. This is why most of these games offer multi-player options!

In theory, the limitations from above could be overcome by allowing the human player to create the necessary cooperative behaviors himself by *programming* them. But programming in the usual Computer Science sense requires a lot of knowledge and to deal with a lot of details and is definitely not something the average player would accept. In this paper, we present another approach to programming behavior that is based on how humans usually explain to other humans their role in a team effort. This is coupled with modeling the behavior of the opponents.

More precisely, we propose to add to games of this type a so-called huddle component (or mode) that can be entered either from savepoints or whenever the player wants. When entered, the huddle allows the player to define a wanted (partial) goal game state that describes the goal the player wants his team of characters to achieve. Like all game states that need to be communicated in the huddle, the partial goal state is entered via a graphical interface that allows the player to place characters where he wants in the world and also to indicate attributes of these characters other than position, for example being dead for an enemy character. In addition to the goal state, the player also has to provide each of his characters with a set of situation-action pairs that are intended to provide the characters with the key ideas of what to do in the current game scenario. This is similar to the drawing board of a sports coach.

To avoid having to go down to the level of programming each individual step for each agent (as is done in the game Frozen Synapse, see [6]), the huddle tries to fill in additional situation-action pairs to achieve the goal using the learning approach from [5], thus generating somewhat "intelligent" characters. Since this requires to simulate the results of applying situation-action pair based strategies to the current game scenario, we have to also model the behaviors of all other characters. While it is possible for a game developer to "cheat" at this by using the real behaviors of these characters, we use the modeling approach using observed situation-action pairs from [3] to reflect exactly what the player has observed, so far, in the game. The time for learning is limited and if the learner does not find a successful extension to the player provided situation-action pairs, the best found team strategy is displayed and the player can provide additional hints (i.e. situation-action pairs). If within the huddle a successful team strategy is found, this strategy can then be applied in the real game, either solving the scenario or revealing additional pieces of the scenario that now need to be dealt with. We instantiate the huddle idea for a simple fantasy themed multi-character quest game and demonstrate the usage of the huddle for 2 different scenarios.

II. Basic definitions and concepts

In this section, we first introduce a very generic definition for agents and multi-agent systems. Then we present an instantiation of this definition for an agent architecture based

Tim Davison and Jörg Denzinger are with the University of Calgary, Canada (email: {tbdaviso,denzinge}@ucalgary.ca)

on prototypical situation-action pairs together with the nearest-neighbor rule, as introduced in [2]. We also present how this architecture can be used to model other agents (as introduced in [4] and extended in [3]). These concepts form the basis of the huddle and will be used in the next section to introduce it.

On a very abstract level, an agent $\mathcal{A}g$ can be seen as a 4-tuple $\mathcal{A}g$=(Sit,Act,Dat,$f_{\mathcal{A}g}$). The set Sit describes the set of situations $\mathcal{A}g$ can be in (as perceived by $\mathcal{A}g$), Act is the set of actions $\mathcal{A}g$ can do, Dat is the set of all possible value combinations of $\mathcal{A}g$'s internal data areas and $f_{\mathcal{A}g} : Sit \times Dat \rightarrow Act$ is the agent's decision function. Internal data areas are all the variables and data structures representing $\mathcal{A}g$'s knowledge. A multi-agent system $\mathcal{M}as$ is a pair $\mathcal{M}as$ = (A,$\mathcal{E}nv$) of a set of agents A and an environment $\mathcal{E}nv$ that the agents in A share.

The agent architecture we are using for the huddle are prototypical (extended) situation-action pairs (SAPs) together with a similarity measure sim on situations applied within the Nearest-Neighbor Rule (NNR) that realizes $f_{\mathcal{A}g}$. More precisely, an element d of the set Dat of our agents consists of a part d_{SAP} and a part $d_{Rest} \in Dat_{Rest}$ (i.e. d = $d_{SAP}d_{Rest}$), where d_{SAP} is a set of pairs (s',a). Here s' denotes an extended situation and $a \in Act$. An extended situation s' consists of a situation $s \in Sit$ and a value $d_{Rest} \in Dat_{Rest}$ (this allows us to bring the current value of Dat into the decision process; usually only some parts of a d_{Rest} are used or no part of such an element at all).

For determining what action to perform in a situation s with Dat-value $d_{SAP}d_{Rest}$, we use a similarity measure sim that measures the similarity between sd_{Rest} and all the extended situations s'_1,...,s'_m in d_{SAP}. If $sim(sd_{Rest}$,$s'_i)$ is maximal, then a_i will be performed (if several extended situations have a maximal similarity, then the one with lowest index is chosen).

For any given agent $\mathcal{A}g$, another agent $\mathcal{A}g_{obs}$ observing this agent's actions starting in a situation s_0 sees the behavior of this agent $B(\mathcal{A}g$,$s_0)$ as a sequence of observed situations and actions, i.e. $B(\mathcal{A}g$,$s_0)$ = s_0,a_0,s_1,a_1,... (where, unfortunately, $s_i \in Sit_{obs}$, i.e. the situations are observed according to $\mathcal{A}g_{obs}$'s observational capabilities). This means that an easy way to model $\mathcal{A}g$ (by $\mathcal{A}g_{obs}$) is to simply take an observed behavior sequence s_0,a_0,s_1,a_1,..., s_k,a_k and convert it into a set of (prototypical) situation-action pairs $\{(s_0,a_0),(s_1,a_1),...,(s_k,a_k)\}$.

It should be noted that in this set there might be two or more pairs that have the same situation but different actions. Then appropriate conflict handling is necessary, as for example taking the newer observation. There are two reasons for this problem in modeling other agents. First, $f_{\mathcal{A}g}$ uses not only the current situation but also the current value of the internal data areas of $\mathcal{A}g$ to make a decision, and $\mathcal{A}g_{obs}$ usually does not know what the current value from Dat is and also not what the values were when it made its observations. This is a general problem in modeling other agents and therefore modeling other agents usually works only for very reactive agents, that do not use Dat much in their decision making. The second problem is that $\mathcal{A}g_{obs}$ might observe situations differently than $\mathcal{A}g$, so that two situations that are different for $\mathcal{A}g$ might be the same for $\mathcal{A}g_{obs}$. Our huddle does not see this as a problem, it will in fact make use of it.

III. The huddle concept

In this section, we will first describe how a computer game having a player control several characters can be seen as a multi-agent system and how a human player gains more and more information about the agents of the game over several attempts to win the game (resp. solve a game scenario). Based on this, we will then describe the huddle formally.

A. A game as a multi-agent system

It is rather obvious that a character in a computer game can be seen as an agent, which makes a game with more than one character a multi-agent system. More precisely, a game $\mathcal{G}ame$ can be seen as $\mathcal{G}ame = (PC \cup NPC, \mathcal{W}orld)$, where PC are the game characters controlled by the player and NPC are all other entities in the game that can perform actions (like, for example, a door that can open or close; but naturally also all characters not controlled by the player), $PC \cap NPC = \emptyset$. In other words, $PC \cup NPC$ is our set A of agents. $\mathcal{W}orld$ is the world that the player (resp. his characters) are put into by the game, the set $\mathcal{E}nv$ in terms of the last section. This means that $\mathcal{W}orld$ can be seen as a set $\mathcal{W}orld = \{e_1, ..., e_i, ...\}$ of environment (world) states and in most games the world is subdivided into a set of scenarios, $\mathcal{W}orld = \mathcal{S}cen_1 \cup ... \cup \mathcal{S}cen_n$, with each scenario $\mathcal{S}cen_i$ using a subset of the world states.

While the developer of a game naturally has a clear idea what $\mathcal{W}orld$ and the $\mathcal{S}cen_i$s are, and also how an agent $npc \in NPC$ is defined (i.e. $npc = (Sit_{npc}, Act_{npc}, Dat_{npc}, f_{npc})$), one of the challenges for a player in such games often is to develop a view of the world and the behaviors of the agents in NPC that is sufficiently near "reality" to allow the player to come up with a strategy for the characters he is controlling to win the game. Furthermore, due to the division into scenarios, this can be subdivided into a sufficiently correct view of the scenario $\mathcal{S}cen_i$ and the elements of NPC occurring in the scenario (NPC_i), to solve the scenario with the group of characters the player controls in the scenario (PC_i).

To develop this view, the player obviously has to play the scenario $\mathcal{S}cen_i$, starting from a world state e_i^0 and creating a trace $tr = (obs(e_i^0), obs(ac_i^0), obs(e_i^1), obs(ac_i^1), ..., obs(ac_i^{k-1}), obs(e_i^k))$ (from the perspective of the game), where ac_i^j is the set of actions taken by all the elements of $NPC_i \cup PC_i$ to create e_i^{j+1}. For an agent $\mathcal{A}g$, $ac_i^j/\mathcal{A}g$ denotes the action taken by $\mathcal{A}g$ in e_i^j. We have to filter this trace by using the function obs, respectively variants of it for $\mathcal{S}cen_i$ and the combined action sets of NPC_i and PC_i, because the player will usually not be able to observe the full world state and all actions performed by the agents in NPC_i (which is usually referred to as "fog of war"). At most, the player will be able to observe what the agents in PC_i are able to observe.

To produce a view that is sufficiently near to what is really happening in the world and the agents in NPC_i, a player usually needs to play the scenario (or parts of it) several times, so that we can characterize the current view νiew of the player as a set of traces, i.e. $\nu iew = (tr_1, ..., tr_s)$.

B. The huddle

As already stated, the vision behind the huddle is to add to a game a component (the *huddle*) that allows the player to take a timeout for the characters he controls, usually after failing a scenario $\mathcal{S}cen_i$ and being back at e_i^0, and to develop a strategy for the agents in PC_i that at least gets them nearer to solving $\mathcal{S}cen_i$. The huddle will be entered with a particular view $\nu iew = (tr_1, ..., tr_s)$, collected for the player by the game, and has as goal developing strategies for the agents in PC_i that can be used in the "real" game. The main tools available in the huddle to achieve this, are

- a strategy editor,
- a game simulator using νiew and
- a cooperative behavior learner.

Using the strategy editor, the player will first create a (partial) situation $\mathcal{G}$ that he considers the goal he wants to achieve. This is for sure a partial world state and the goal is considered to be reached whenever the game simulator gets into a state that matches it (which we define as all features that are explicitly set in $\mathcal{G}$ having identical values in this state).

After that, the strategy editor has to allow the player to enter for each of his characters $pc_j \in PC_i$ a set of situation-action pairs $Sap_j^{pl} = \{sap_{j,1}^{pl}, ..., sap_{j,n_j}^{pl}\}$, with $sap_{j,l}^{pl} = (s_{j,l}^{pl}, a_{j,l}^{pl})$ and $s_{j,l}^{pl} \in Sit_{pc_j}$ and $a_{j,l}^{pl} \in Act_{pc_j}$. The interface for this should resemble the interface the game provides, with the additional possibility to change features of the world, the characters from NPC_i that have already occurred in νiew and the characters from PC_i (including the features of the character for which a situation is created, which makes it an extended situation). To allow for that, the game developer needs to define what features are accessible, essentially creating a set $\mathcal{W}orld_{hudd}(\nu iew)$ that contains all possible states of the huddle's current view of the game world. While in theory we could just set $\mathcal{W}orld_{hudd}(\nu iew) = \mathcal{W}orld$, this could give away information about the game the player should not have at a given point (i.e. given the players current view of the game). Instead, a (huddle) world state $e_{hudd,k}$ should be only a partial real world state, leaving out features of the landscape that have not been seen in any of the traces in νiew (after application of *obs*), as well as features of the agents in NPC_i that have not been seen in those traces. Since this means that $\mathcal{W}orld_{hudd}$ will dynamically change with the experiences of the player, we have the player's view as a parameter of it. Naturally, the game designer can also choose to reveal some information, for example for inexperienced players.

As stated before, the Sap_j^{pl}s should only provide the key ideas for the strategy that the player wants the agents in PC_i to use to fulfill $\mathcal{G}$. Filling in the gaps in form of additional sets Sap_j^{ln} of SAPs is the task of the learner for cooperative behavior. We use an evolutionary learner, as suggested in [5]. This learner works on sets of strategies (individuals), where each strategy consists of a set of SAPs for each of the agents in PC_i. While the initial set of individuals, the start population, is generated randomly, the following sets (populations) are created by replacing the worst individuals of the previous population with new individuals created using so-called genetic operators. A *crossover* takes two individuals as "parents" and creates a new individual by randomly selecting SAPs from the parents for each of the SAP-sets in an individual (i.e. for each agent in PC_i). A *mutation* uses one parent and either substitutes one SAP in one of the sets of the parent by a randomly created pair to produce the new individual, or it deletes a randomly chosen SAP or adds a randomly created SAP.

A key component of this evolutionary learning approach is how the quality of a strategy is determined, i.e. the so-called fitness function. As suggested in [5], the fitness of an individual will be determined using the already mentioned game simulator by trying out the strategy represented by the individual together with the player-provided SAPs for the agents in the Sap_j^{pl}s in a game simulation. This simulation will start at e_i^0 and goes on for either a given number of game states or until either a game state matching $\mathcal{G}$ is reached or there are no actions for any agent in PC_i anymore. We are talking about a simulation, despite the fact that we obviously could just use the real game, for the same reason why we introduced $\mathcal{W}orld_{hudd}(\nu iew)$: part of the fun and/or challenge for a player is making experiences with the game and figuring out solutions for the obstacles the experiences reveal. Naturally, a game designer will make sure that as much code as possible can be shared between the game and the huddle and there might be even parts of the simulator that could reveal more about the game than what the player has experienced so far. But at the core, the simulator must reflect $\mathcal{W}orld_{hudd}(\nu iew)$ and its analog for the NPCs: $NPC_{i,hudd}(\nu iew)$.

So, for an $npc \in NPC_i$, we should not simply use its game implementation, i.e. $(Sit_{npc}, Act_{npc}, Dat_{npc}, f_{npc})$, instead we need to create for the simulator an npc_{hudd} ($\in NPC_{i,hudd}$) that is the model of npc based on what the player could have learned about npc in the traces of νiew. Obviously, the structure of Dat_{npc}, the definition of f_{npc} and even an idea of what npc can perceive (i.e. the definition of Sit_{npc}), are very difficult to guess, but what we need is just some idea of the behavior of npc and as long as npc_{hudd} comes near to reality the huddle will work. Therefore we use the modeling approach using SAPs sketched in Section II: we assume that $Sit_{npc_{hudd}}$ represents the perceptions of the player, $Act_{npc_{hudd}}$ contains the actions npc performed in the traces in νiew, i.e. $Act_{npc_{hudd}} = \bigcup_{tr \in \nu iew} \bigcup_{ac_i^l \in tr} obs(ac_i^l)/npc$, $f_{npc_{hudd}}$ realizes the action selection based on prototypical SAPs and the NNR, and the current value of $Dat_{npc_{hudd}}$ contains (nearly) all the SAPs for npc observed in the traces in νiew (i.e. $\bigcup_{tr \in \nu iew} \bigcup_{l=0}^{k} (obs(e_i^l), obs(ac_i^l)/npc)$). The "nearly" in the last sentence is due to wanting to have only at most one SAP for each situation. Therefore, whenever a case with more than one occurs, we can either automatically put into $Dat_{npc_{hudd}}$ only the latest pair from the newest trace or have the player decide (we chose the first variant for our evaluations).

Coming back to the fitness function, performing a simulation for a strategy for the PC_is allows us to measure how near the strategy came to the goal situation $\mathcal{G}$. While naturally the concrete definition of the function will depend on the particular game, we suggest evaluating each occurring world state with regard to nearness to $\mathcal{G}$ and to add these evaluations up. Also, the same ideas as for the similarity measure for situations used by the SAP-based agent architecture can be used to define the nearness. Naturally, since the game designer has to provide the similarity measure and the fitness function, it is possible to test the appropriateness of these functions before the release of a game.

The whole learning of the Sap_j^{ln}s to complete the Sap_j^{pl}s naturally can not take too much time, since the player will want to see if the strategy really works. Limiting the steps in the simulation helps with this, but also limiting the number of populations that the learner goes through and the number of new individuals created for each population. Due to this, it can happen that no individual getting to $\mathcal{G}$ is found. In this case, the simulator allows the player to observe the performance of the best found individual and to adjust the Sap_j^{pl}s he has given the system. Then the learning can be repeated, starting either with randomly created SAPs, again, or starting with a learned strategy from the last round indicated by the player and adding SAPs to that strategy.

If an individual is found that reaches $\mathcal{G}$, the task of the huddle is mostly done. For each PC_i, the set $Sap_j^{ln} \cup Sap_j^{pl}$ is made available to the real game, so that the player can choose to have the behavior of the PC_Is in the main game replaced by an agent based on using SAPs and the NNR, naturally using the found SAP sets from the huddle.

IV. Applying the huddle

In this section, we apply the huddle concept from the last section to a concrete game, namely a dungeons and dragons like role playing game where the player is responsible for guiding a group of characters traveling through a sequence of dungeon caverns and having to fight monsters to pass each individual cavern. While we would have preferred to use an existing game, the huddle requires a rather tight coupling with the game, so that we need access to the complete source code, which was not possible for the interesting commercial games for obvious reasons.

In the scenarios in the next section, the player is responsible for 3 characters from 3 different classes, a mage, a warrior and a priest (these 3 characters form PC, but also the PC_i for all scenarios). All these characters have in common that they have a $Hlth$ attribute that represents the current health of the character and that they have a current position pos. Additionally, each of the classes has individual abilities that can be invoked as an action. These abilities are for the priest Heal and HoT (heal over time), which either restores a larger amount of health to another character in one action or smaller amounts of health over several seconds, for the mage Fireball and Rain of Fire (attacking either a single enemy or a group of enemies) and for the warrior Taunt (to get enemies to concentrate on the warrior) and Sword Attack (which is the attack action for this class). All these actions come with different cool down periods, i.e. the time until the action can be applied again, and ”cast times”, i.e. the time needed to perform the action.

For NPC, we have a minion class and a boss character. A character in the minions class is very similar to the mage character, except that it can only cast Fireball. The boss has two abilities, Sword Attack and Buff Sword. The Sword attack is the same as the warrior's Sword Attack, however it deals more damage. The Buff Sword ability can only be used when this character is at 75%, 50%, and 25% of its maximum $Hlth$ and causes the next Sword Attack to deal a large amount of damage to all characters in a large radius around the boss. The mage and the priest cannot survive this damage. All characters in NPC in the real game use for targeting the concept of threat (which is well known from games like World of Warcraft, see [1]) that is trying to measure the threat of a character in PC for them, based on the actions this character has taken in the past. In general, most actions of a character in PC create medium threat, except for Fireball and Sword Attack that have a low threat and Taunt that produces, as the name suggests, a high threat. The NPC character will always attack the highest threat PC character (when using an attack action that requires a target). If this PC character is out of range the NPC character will move towards it. A state in $\mathcal{W}orld$ does not provide much more information than the positions and other features of all elements in PC and NPC (which means that $\mathcal{W}orld$ itself is only adding the map data to the agent information). Consequently, $\mathcal{W}orld = \mathcal{W}orld_{hudd}(\nu iew)$ for all νiew.

In the huddle, the strategy editor allows the player to enter a situation by moving the characters in PC and the appropriate NPC_i to where they should be in the situation that is defined and to provide for each of the characters in the scenario values for their health. We are also offering for the elements of NPC_i additional features that might be relevant, like the color of parts of a character, see the next section, or irrelevant for the game (to make the relevant features not too obvious and hence giving away things). For the action of an SAP, the player can choose for the character it is providing the SAP for from a dropdown menu, either a movement to a particular position or the usage of an ability with the target for the ability (if one is needed). As mentioned in the last section, the player uses this functionality to enter Sap_h^{pl} for each $h \in PC$.

The similarity measure sim used for applying the NNR to a set of SAPs is summing up individual similarity measures for each element in the situation of an SAP. As already stated, we have for each character h in $PC \cup NPC$ the position pos_h and we have its health $Hlth_h$. In the next section, we only use two additional features of the boss character, namely the body color info $colb_{Boss}$ and the sword color info $cols_{Boss}$. Then, for two situations s_1 and s_2, we have

$$
\begin{aligned}
sim(s_1, s_2) = \sum_{h \in PC \cup NPC} & (w_{h,pos} \times sim_p(pos_{h,1}, pos_{h,2}) \\
& + w_{h,Hlth} \times sim_{Hlth}(Hlth_{h,1}, Hlth_{h,2})) \\
& + w_{boss,colb} \times sim_{colb}(colb_{Boss,1}, colb_{Boss,2}) \\
& + w_{boss,cols} \times sim_{cols}(cols_{Boss,1}, cols_{Boss,2})
\end{aligned}
$$

Here, the different ws indicate weight parameters that can be used by the game developer to produce behaviors that require more or less guidance by the player. We will see this in the next section. The ws also allow us to instantiate sim differently for usage by the strategies for the characters in PC and for usage as an npc_{hudd} for an $npc \in NPC$ in the real game.

The different sub-similarity measures are usually just the difference of the numerical values (for sim_p of the two coordinates and for sim_{Hlth} of the current number). The similarity of the two boss colors is either 0, if they are the same, and 1, if they are different.

For our evolutionary learner, we already described the general genetic operators in the last section. The mutation that replaces a SAP is realized by going through the SAP and replacing each value in the situation vector with a different one with a given probability r_{mut}. It also changes the action in the SAP with this probability. The parents for a genetic operator application are selected using fitness proportionate selection (i.e. individuals with a better fitness have a higher probability to be selected).

The fitness measure used by the learner is rather simple: we compute for each huddle game state e_i^j encountered in the simulation what the difference in $Hlth$ to the last state e_i^{j-1} is for each of the characters and sum up the differences for the characters in PC and for the characters in NPC_i and subtract the two values from each other (and, as already stated, sum this up over all huddle game states encountered in the simulation). This reflects the fact that the goal in all scenarios is to kill all the characters in NPC_i (the player might re-define the goal using the strategy editor, but killing all NPC_is is the default).

Finally, the modeling of the characters in NPC_i is performed exactly as presented in the last section, with conflicts (i.e. having more than one pair for the same situation) resolved by using the latest observation.

V. Usage scenarios

To illustrate the utility of the huddle, we have designed two scenarios, each presenting the player with a puzzle that is an intellectual challenge and that would require skill with the game's controls to complete. For these scenarios, we provide examples of how the player can use the huddle. In the examples, the player will first try the game using the manual controls, and after a few failed attempts, he will enter the huddle to define skeleton strategies that are then elaborated upon by the GA. When the GA finds a potential solution, it will give the player a chance to try it, and modify it if necessary. If a satisfactory solution is not found, the player can invoke the GA again.

In both examples, the GA is configured with a population size $n_{pop} = 20$ and the mutation rate is $r_{mut} = 0.2$. The maximum number of SAPs per agent is $n_{sapMax} = 12$ and the initial number of SAPs per agent is $n_{sapIni} = 8$. Finally, the maximum number of generations before the GA declares that it could not find a solution is 50.

In our first scenario, the player controls three characters: a warrior, a priest and a mage, with the abilities outlined in the previous section. The enemies consist of four identical minions placed so as to block the player's access to the next dungeon, see Figure 1. Attacking any one of the minions causes all of the minions to attack the character that initiated the attack. The weights used in the definition of sim were set as follows: For the priest pri, we have $w_{pri,pos} = 0$ (to have the priest focus on healing) and $w_{pri,Hlth} = 1$. For the mage mag and warrior war we have $w_{mag,pos} = w_{war,pos} = w_{mag,Hlth} = w_{war,Hlth} = 1$. For the minions, we have for both the position and the health a weight of 1. And, naturally, without the boss, we do not need the color features, so $w_{boss,colb} = w_{boss,cols} = 0$

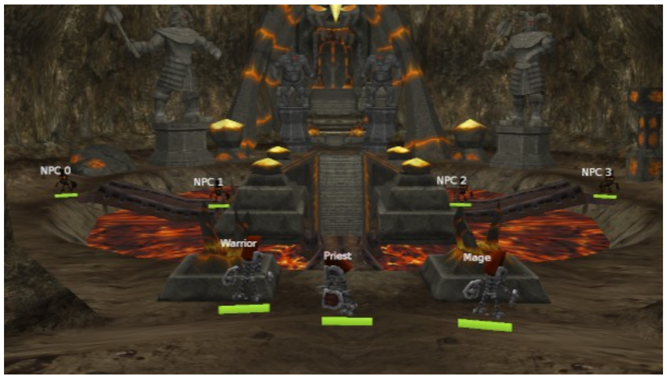

Fig. 1. Scenario 1: Initial Configuration. The enemy minions are labeled NPC 0, 1, 2 and 3. The player controls the Warrior, Priest and Mage.

This particular puzzle is designed so that the mage character has to use his Rain of Fire area-of-effect ability on a group of clustered minions, while they are focused on the warrior. If the mage does not strike more than one character with the area-of-effect ability, then the battle will last too long, and the priest will not be able to keep up with the damage being dealt. Likewise, the player must try to keep as many minions as possible focused on the warrior by using the Taunt ability, otherwise, there is once again too much damage for the priest to heal (recall that the warrior takes reduced damage).

When the player first attempts this scenario, he has manual control over the characters in his party. Unless the player is particularly skillful, he will have a hard time coordinating all of the characters, so this is a good opportunity to apply the game's huddle component.

The huddle interface is an extension of the real game interface. The current position of the PCs and the NPCs are represented by their current 3D models, while the agents referenced in the SAPs are represented by color coded cylinders. The blue cylinder corresponds to the agent for which an SAP is being defined, while red and white cylinders denote, respectively, enemy and friendly characters. An arrow indicates the target of an action, if the target of an action is the character for which an SAP is being defined, there is no arrow. Figure 2 contains two examples of the huddle interface for a strategy that we will define next.

One possible strategy that we designed to illustrate the huddle concept begins by having the warrior character taunt the first three minions, in the order NPC 2, 0 and then 1 (Figure 2 illustrates the taunts for NPC 0 and 1). Meanwhile, to mitigate some of the damage being done to the warrior, and to keep damage off of the priest, the mage attacks NPC 3. We also have the priest focus his heals on the warrior. By trying out his skeleton strategy in the real game, the player helps to improve the observed model of the minions that will be used by the learner. Once the player has realized that this strategy is not enough, he invokes the GA.

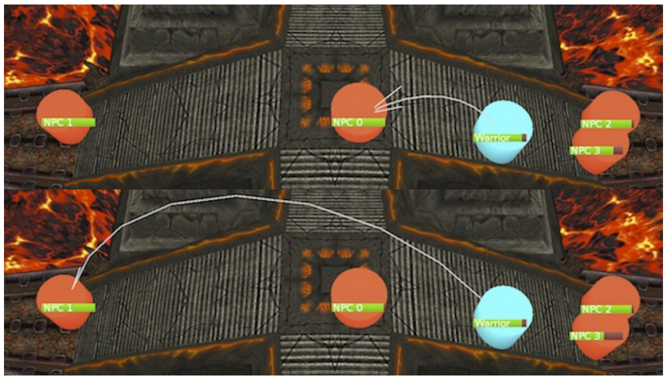

Fig. 2. Scenario 1: In the top portion of this figure, the warrior taunts NPC 0, while in the bottom half, the warrior taunts NPC 1. Notice the subtle difference in the healths of NPC 0 and 3 between the two, this is the primary difference as seen by sim. The other agent positions are not shown.

For this example, after 9 generations, the GA found a number of solutions that beat the observed model of the minions. However, the player's task is not complete, as the solution does not work as well in the real game. In particular, the mage is using his Rain of Fire ability too early. To prevent this, the player has to specify two additional SAPs for the mage. The first casts the Rain of Fire ability when the minions are clustered, the second casts a Fireball just before the minions have gathered. By casting Fireball, just before the minions have gathered, we ensure that the similarity measure chooses SAPs that don't waste the Rain of Fire ability (which has a long cool-down). Figure 3 illustrates this.

The other problem with the solution that the GA found is that the warrior is not keeping the minions focused on him. The player can solve this problem by making the warrior attack NPCs 0 through 2 in turn in a fashion similar to that of Figure 2. These modifications represent a complete strategy, so that the GA terminates with the first individual it evaluates (which contains all the GA produced SAPs from the player's first attempt).

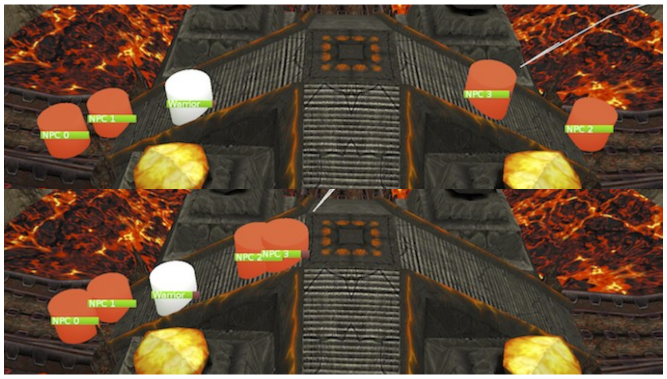

Fig. 3. Scenario 1: In the top portion of this figure, the mage will cast fireballs at minion 3, when the minions are clustered close together, then the mage will cast his Rain of Fire area-of-effect ability on the group, as shown in the bottom half. The other agent positions are not shown.

The combined behavior for the mage contains 10 SAPs, two of which came from the player. Two of the GA's SAPs were not used, the rest were focused on killing NPC 3, and finishing off the remaining minions towards the end of the encounter. The priest used the maximum n_{sapMax} of 12 SAPs, two of which came from the player. The GA's SAPs mainly focused on healing the warrior, especially towards the start of the scenario. By the end of the scenario when there was less damage being directed at the warrior the priest was also healing itself and the mage. One particularly interesting SAP, that the GA came up with, had the priest move back to a central location after each cast. For the agents on top of the pyramid structure in the center of the scenario, this blocks some of the minion's attacks, forcing them to walk towards the priest until they had line-of-sight. The warrior had the simplest strategy, with only 7 SAPs (four from the player). One of the GA's SAPs was not used. The GA learned to use sword-attacks, instead of taunts, once the warrior had the attention of the minions. Sword attacks deal more damage than taunts, but generate less threat.

In the second scenario, the player controls the same three characters as in the first scenario. Instead of several minions, there is the single boss character whose abilities are described in the previous section. The initial state of the scenario is as in Figure 4. This scenario is designed to show how the puzzle aspects of gameplay can be enhanced by the huddle. Particularly, we employ a character with a devastating attack, and a subtle tell that indicates the attack is about to happen. If the player can puzzle out the tell, then he can move his weaker characters to safety just before the devastating attack reaches them. The weights used in the definition of sim were set as follows: For the priest pri, we have $w_{pri,pos} = 0$ (to have the priest focus on healing, again) and $w_{pri,Hlth} = 1$. For the mage mag and warrior war we have $w_{mag,pos} = w_{war,pos} = w_{mag,Hlth} = w_{war,Hlth} = 1$, again. For the boss $boss$, we have $w_{boss,pos} = 1$, $w_{boss,Hlth} = 2$, $w_{boss,colb} = 0$ (since the body color is irrelevant, see below) and $w_{boss,cols} = 3$.

The puzzle we have prepared for the player is based upon

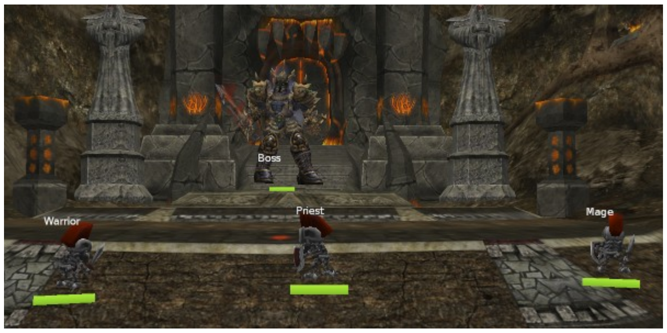

Fig. 4. Scenario 2: Initial configuration.

colors of different parts of the boss character: the player must recognize what combination of colors serves as an indicator that the boss is about to perform his powerful attack. In particular, both the sword and body glow different colors that change at random intervals. The body color is designed to confuse the player. The sword color normally glows either blue or colorless. When the boss performs the Buff Sword ability, the sword glows white for one attack, and then white again for the next powerful attack. The solution to this puzzle is to recognize that the white sword color is the tell.

When the player first attempts the scenario, he has manual control over the characters in his party. After a few failed attempts, he might grow weary of repeating the same actions, and instead invokes the huddle to create a simple strategy that attacks the boss so that he can focus on a way to beat him. One such strategy has the priest start by giving a HoT to the warrior, the warrior using his Taunt on the boss, while the mage casts Fireball. While defining the mage's Fireball SAP (Figure 5), he notices two menu options for the sword and body color, giving him a hint that these are important clues. At this point the player should realize that there are no abilities that his characters posses that can stop the powerful attack, the warrior can survive the attack but the mage and priest cannot, and that the arena contains pillars. A keen player might realize that if he can divine the tell for the Boss' powerful attack then maybe he can hide his characters behind the pillar. So, he defines two additional SAPs that move the mage and priest to safety for some random combination of the Boss' sword and body colors. After several additional attempts, the player realizes that the white-sword is the tell, and modifies the SAPs that move the priest and mage behind a pillar accordingly, these SAPs can be seen in Figure 6. Now the player's characters can survive the powerful attack, but there are not enough SAPs to beat the scenario, so the player starts the GA to fill in the rest of the strategy.

For this example, the GA found a solution that did not work in the real game after 8 generations. The priest's behavior contained 8 SAPs, 6 of which came from the GA. These SAPs were focused on healing the warrior, but one in particular kept the priest between the warrior and the pillar. When the boss casted his powerful attack, the priest was still in line-of-sight and was killed. The mage's behavior consisted of 8 SAPs, two of which were defined by the player. Only the player's SAPs were used in the real game. Finally, the warrior's behavior contained 9 SAPs, one of which was defined by the player. The only SAP used by the warrior was his Taunt ability.

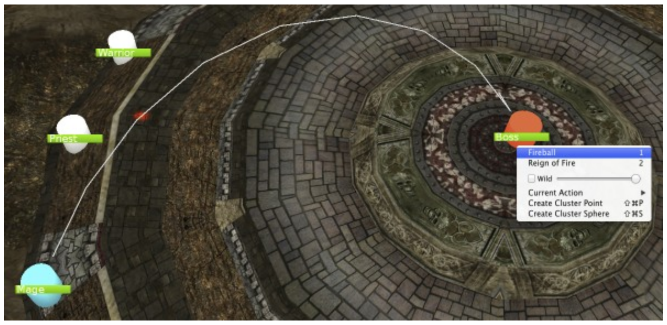

Fig. 5. Scenario 2: The initial strategy for the mage in Scenario 2. The mage, priest and warrior, and the boss locations are represented by the blue, white and red cylinders respectively. The menu shows that the Fireball will be cast against the Boss and provides options to specify the colors of the Boss' sword and body.

Observing the results produced by the GA, the player should realize that the boss should be positioned closer to the pillars. So, the player discards the strategy produced by the GA, and with the strategy that he previously gave to the GA, defines two additional SAPs for the warrior. The first SAP moves the warrior between the pillars after his first Taunt and the second SAP Taunts the Boss once he is near the pillars. Once again, the player invokes the GA to fill in the details (such as the healing behavior of the priest, or when the warrior and mage should use their other abilities).

In our example, the GA found a solution based upon the player's improved strategy after only 5 generations. The warrior's behavior consisted of 6 SAPs, three defined by the player, only the player defined SAPs were used. The priest's behavior consisted of 8 SAPs, two defined by the player. Five of the SAPs were used, three from the GA. The mage's behavior included 8 SAPs, two defined by the player. Four of the GA's SAPs were not used. One interesting SAP that the GA came up with had the mage cast his Rain of Fire ability after the second buffed boss attack. This ability takes 15 seconds to cast, cannot be stopped, and is more powerful than the Fireball ability. During that time the mage is vulnerable to the boss's Buff Sword attack, but by using the ability right after one of those attacks the mage can maximize his damage.

Many of the behaviors produced by the GA have a number of SAPs that are not used. This is because the GA is finding solutions after relatively few generations, and as soon as a solution is found, it reports that solution to the player without having the opportunity to remove extraneous SAPs. This naturally raises the question if the learner in the huddle is consistently finding solutions.

To look into this, we saved the second strategy defined by the player for scenario 2, and used it as the basis for evaluating the performance of the GA. This skeleton strategy is missing a few important SAPs to heal the warrior, and SAPs that increase

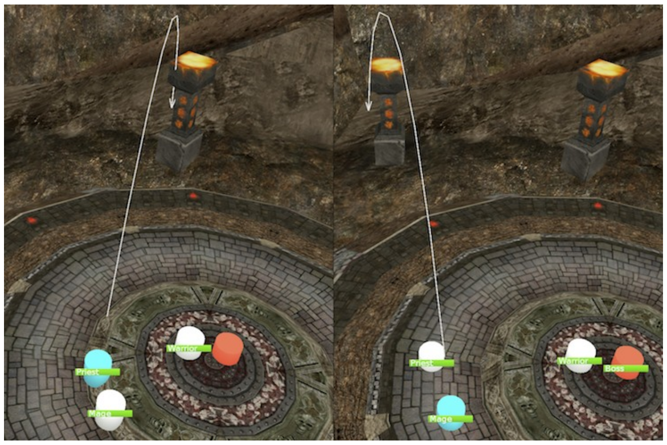

Fig. 6. Scenario 2: On the left, a SAP that moves the priest behind cover. On the right, a SAP that moves the mage behind cover.

the damage that the warrior and the mage are dealing to the boss. We performed 20 runs of the learner, and noted the generation of the first solution reported by the GA that also worked in the real game. The least number of generations was only 5 (this strategy was described earlier), while the most number of generations was 24. On average the GA found a solution within 10 generations (with a standard deviation of 4.56). While this represents quite a variety, nevertheless in all the learning runs a solution was found, showing that the learner is doing the job required of it.

VI. Related work

As already stated, nearly every game where a player is responsible for a group of characters allows the player to choose between several pre-defined behaviors for a particular character. But there are only a few games that allow the player to customize the behavior of each of the characters under their control such that all of the characters act at once. Like the huddle, these games employ an interface for defining that behavior that is different from the real game's interface. Frozen Synapse (see [6]) is a tactical squad-based game that allows the player to define action sequences for each of the characters under his control in an action-editor mode. These sequences are then played out in short time-intervals in the real game against the opponent. The huddle differs from this game in that it has a learning component that allows the player to focus on overall strategy instead of every detail necessary to win a given scenario, and that a small number of SAPs can achieve the same result as a long sequence of actions because a single SAP can be applied to many different situations.

The NERO approach (see [8]), in contrast, focuses on learning general behaviors for the characters the player is responsible for without allowing the player to suggest parts of the behavior. The architecture used for characters is a neural network, which naturally makes it nearly impossible to integrate direct advice by the player. Instead the player needs to create scenarios that try to teach what he wants the characters to do (see [7]).

VII. Conclusion and future work

We presented the huddle, an extension for games where the human player is responsible for nearly simultaneously guiding a group of characters through different scenarios. By allowing the player to provide his characters with the important ideas of how to handle a scenario and having a machine learner fill in the blanks in the resulting strategies for the characters, every game of this type can be made attractive for players that prefer to solve problems and are not much interested in artistry with the controller. We demonstrated the usefulness of the huddle with two usage scenarios around a role playing game, highlighting the need by the learner for simulating the game according to what the player has observed so far, to avoid giving away the challenge of a scenario.

We think that the demonstrated use of the huddle is just a beginning in exploring the potential of the concept. Having a game designer being able to create challenges for the player's group of characters that require guided, highly simultaneous actions is obviously an interesting extension to what can be put into a game. But the components of the huddle offer more interesting possibilities for a game designer. For example, the similarity measure used by the nearest-neighbor rule and the fitness measure for the learner can be manipulated to help the player or to make it more difficult to create a strategy for the characters, helping with creating different difficulty levels for players (i.e. having a more or less ”intelligent” group of characters). Exploring these possibilities is one of our goals in the future. We also see potential in the huddle idea for serious games aimed at helping a player to develop team planing and leadership skills.

References

[1] Blizzard Entertainment, ”World of Warcraft”, v. 4.3.3, 2012. http://us.battle.net/wow/en/, accessed Apr. 10, 2012.

[2] J. Denzinger, M. Fuchs, ”Experiments in Learning Prototypical Situations for Variants of the Pursuit Game”, in *Proc. ICMAS-96*, 1996, pp. 48–55.

[3] J. Denzinger, J. Hamdan, ”Improving observation-based modeling of other agents using tentative stereotyping and compactification through kd-tree structuring”, *WIAS 4(3)*, pp. 255–270, 2006.

[4] J. Denzinger, M. Kordt, ”Evolutionary On-Line Learning of Cooperative Behavior with Situation-Action Pairs”, in *Proc. ICMAS-00*, 2000, pp. 103–110.

[5] J. Denzinger, C. Winder, ”Combining coaching and learning to create cooperative character behavior”, in *Proc. CIG-05*, 2005, pp. 78–85.

[6] ”Frozen Synapse”, http://www.frozensynapse.com/index.html, accessed Apr. 10, 2012.

[7] I. Karpov, V. Valsalam, R. Miikkulainen ”Human-Assisted Neuroevolution Through Shaping, Advice and Examples”, in *Proc. GECCO*, 2011, pp. 371–378.

[8] K. Stanley, B. Bryant, R. Miikkulainen, ”Evolving Neural Network Agents in the NERO Video Game”, in *Proc. CIG-05*, 2005, pp. 182–189.