Heuristics for Sleep and Heal in Combat

Shuo Xu
School of Computer Science
McGill University
Montréal, Québec, Canada
shuo.xu@mail.mcgill.ca

Clark Verbrugge
School of Computer Science
McGill University
Montréal, Québec, Canada
clump@cs.mcgill.ca

***Abstract*—Basic attack and defense actions in games are often extended by more powerful actions, including the ability to temporarily incapacitate an enemy through sleep or stun, the ability to restore health through healing, and others. Use of these abilities can have a dramatic impact on combat outcome, and so is typically strongly limited. This implies a non-trivial decision process, and for an AI to effectively use these actions it must consider the potential benefit, opportunity cost, and the complexity of choosing an appropriate target. In this work we develop a formal model to explore optimized use of sleep and heal in small-scale combat scenarios. We consider different heuristics that can guide the use of such actions; experimental work based on *Pokémon* combats shows that significant improvements are possible over the basic, greedy strategies commonly employed by AI agents. Our work allows for better performance by companion and enemy AIs, and also gives guidance to game designers looking to incorporate advanced combat actions without overly unbalancing combat.**

Introduction

Multi-agent combat in Role Playing Games (RPGs) is commonly supplemented by powerful abilities, such as sleep or heal, which allow a team to improve their combat chances by (temporarily) disabling an opponent, or by saving an ally from potential death. Unbounded, these abilities can easily trivialize combat and so most games also impose heavy constraints on their use, making the choice of if, when, and on whom to use such an ability a non-trivial part of the game complexity, and one of the skills players must learn and optimize through multiple battles and repeated gameplay. Non-player characters (NPCs) on both the player side and enemy side, however, can also wield these abilities, and while hand-scripted or randomized approaches are commonly used for their speed and simplicity, avoiding the need for expensive search in action selection, the resulting choices do not always meet player expectation of intelligent companions or opponents.

In this work we develop a formal cost-benefit model to represent the impact of sleep and heal in small-scale game combats. We use this analysis to develop simple and efficient heuristics for selecting whether to use sleep or heal instead of attack actions, considering sleep and heal separately as well as in combination. In each case we validate our heuristics through detailed experimental analysis of abstract combat scenarios based on *Pokémon*. Measurements on different performance factors, including win-rate, remaining team health, and total damage dealt show significant improvements over other, common heuristics, including *Pokémon*'s scripted design approach. Specific contributions of our work include:

- We perform a formal analysis of sleep and heal actions, developing a cost-benefit model that we then use to define an optimizing heuristic for action selection and targeting. As far as we know ours is the first work focused on sleep and heal in combat games.
- Our design is backed by significant experimental work based on *Pokémon* combat and character attributes, showing that our heuristics are manifestly better than common, basic approaches.
- A unified cost-benefit model for both sleep and heal allows us to combine them into a single heuristic, which we evaluate in larger scale situations.

Related Work

Team combats in RPGs are essentially (small-scale) *attrition games,* where one team must fully eliminate another. These are already known to be computationally complex, even with just basic attack and defense. Furtak and Buro, for instance, present proofs on the complexity of two-player attrition games showing that the problem is computationally hard for most game cases, and deciding the existence of deterministic winning strategies for basic attrition games is PSPACE-hard and in EXPTIME [1]. Ontañón *et al.* provide a survey of existing works on solving AI problems in the commercial game *StarCraft*—a much larger scale example of an attrition game compared to the ones we consider [2]. They discuss topics such as current tactics, strategies, and state-of-art AI agents for StarCraft, shedding light on challenges in general attrition games. In smaller contexts, Tremblay *et al.* proposed a greedy heuristic for enemy targeting based on enemy *threat,* a value positively related to enemy attack strength, and negatively to health [3]. The theoretical justification for that heuristic was validated in realistic, FPS-inspired combat scenarios, but still only allows for attack as a combat action.

Combat AIs based on searching through a decision or state space trade runtime performance for improved behaviour that can better adapt to dynamic contexts. Brute-force (DFS or BFS) search does not scale well of course, but can be improved through use of alpha-beta or one of its variants. Work on Fast Alpha-Beta Search, for instance, shows that a search approach can perform better than many scripted strategies in RTS games, and can meet real-time constraints for small scenarios [5].

Even when limited to just attack (or move) scenarios, however, search approaches are necessarily non-exhaustive, truncating search depth to meet timing requirements, and so dependent on good heuristics for state evaluation. Stanescu *et al.* considered the use of Lanchester models, a military approach to estimating combat losses suitable for large army interactions [6]. They use this to improve accuracy of state estimation, significantly improving a search-based StarCraft bot.

More efficient search can also be performed using heuristic search algorithms, such as in Monte Carlo Tree Search (MCTS), a search algorithm that relies on random sampling [7]. Bruce Abramson first experimented with the idea in turn-based two-player games including Tic-tac-toe and Chess [8]. The MCTS algorithm has since been extended to solve AI problems in more genres of modern computer games including real-time games such as *Total War: Rome II* [9], card games such as *Magic: The Gathering* [10], *etc.* MCTS can be complicated to implement in combat games, as state estimates are based on simulating playout, and thus can be highly approximate. Uriarte and Ontañón proposed a combat model for 2-person attrition games, aiming to define, and learn, an accurate *forward model* for state transitions in MCTS search [11], which can then be applied to StarCraft. A paper by Browne *et al.* summarizes recent work on the MCTS algorithm itself [12].

Game research has also explored use of the Rapidly exploring Random Tree (RRT) algorithm for rapid, search-based analysis. RRT was first introduced by LaValle in 1998 to solve path-finding problems [13]. Bruce then adapted it to the sampling-based planning algorithm for discrete space problems in his thesis in 2004 [14], and it has since found use in several game AI systems, including ones aimed at platformers [15], [16], and stealth games [17]. RRT has advantages in flexibility, but practical use in combat analysis has not yet been shown sufficiently effective or efficient [18].

Combat Model

Our approach relies on a basic, formal model of combat. We also consider multiple forms of evaluation, as combat success is not strictly boolean in practice. Below we motivate and describe our design for both aspects, followed by a detailed description of the combat parametrization.

Model

Although sleep and heal are common, near ubiquitous features of RPG combat, there exists enormous variation in how these skills may be incorporated into combat. The ability to use a sleep action, for example, may be unique to one character, or based on character type or acquired abilities, and thus available to multiple agents. The effect itself may apply to only a single, targeted enemy, or a range of enemies based on area or proximity, with the potential to affect agents on the same side too, as friendly fire. It can have different durations and durative properties, especially in terms of whether sleeping characters can or cannot be woken by attacks, and various casting costs as well, invoked as part of a set of actions from which selection can be done without replacement, or based on character resources, such as mana or an inventory of objects (such as scrolls). Healing has similar complexity in parametrization. In order to make progress in formal modeling, we thus commit to a single, simple formulation, which while perhaps not fully general, allows for concrete results, and can be extended to variant designs.

Abstractly, we assume a turn-based, 2-team attrition game, with agents on one side termed players, and the other side enemies. Each agent has a (static) attack value, a maximum health, and set of possible actions, which minimally includes attack (which defaults to targeting the enemy with highest *threat* [3]), and may include heal and/or sleep, with use of the latter two constrained by a resource cost and initial supply, and for which we will consider multiple targeting heuristics. Each agent also has a state, either healthy, dead, or sleeping. In the latter case, a turn-counter keeps track of the remaining sleep duration, and we disallow sleep actions on a sleeping agent, as the ability to stack sleeps mainly mimics a sleep of longer duration. We do not model occlusion or geometry, and assume deterministic results from actions; this eliminates the probabilistic element, which affects the decision process of course, but does not change the underlying basis for the decision.

Evaluation

The success of team combat of the sort found in RPG and action games can be measured in different ways. Abstract attrition games usually focus on "last person standing" as a binary measure of success, but in real games, and in comparing effectiveness of different combat choices, this is not always sufficient. Remaining health is an important criterion as well, as health restoration is not always instantaneous or free, and so impacts subsequent combats. Human players may also be strongly invested in (or entirely embodied by) one character in their team, and thus the survival and health of a primary agent may be paramount. Symmetrically, it is not always possible for player teams to fully eliminate the enemy team. For example, in "boss fights" of *World of Warcraft* the enemy leader may not be supposed to be killed. In these situations the goal is to do as much damage as possible before players die or within some time period.

We thus consider three forms of success in evaluating combat scenarios, measured over multiple simulations of the same situation:

- **WIN:** the ratio of combat wins for the player team, expressed as a percentage.
- **HEALTH:** the average sum of player team health, adding up the remaining health of each surviving player team member after each combat terminates. Note that dead characters will have 0 health.
- **DAMAGE:** the average sum of damage done to enemies, adding up the difference between starting and remaining health of each enemy team member after each combat terminates. This is primarily aimed at evaluating simulations where enemies are unbeatable.

Combat Context and Parametrization

Our model is sufficient to approximate many games. We base our specific combat implementation and scale our attack, health and other attributes based on values from the *Pokémon* game, a well known RPG consisting of turn-based, team-based (1–3 agents/side, although we consider larger combats as well) combat between different classes of creatures—"pokémons" (pocket monsters). The game includes a total of 721 different pokémons (as of 2015) [19], giving us a large set of varied agents for simulation. Note that we do not model all attributes of pokémons, as *defense, speed,* and the special attack/defense values complicate damage calculations, without changing the overall process. We do, however, include *Power Points* (PP), a resource which limits the number of each kind of action that can be performed, with sleep and heal much more heavily constrained than attack (which is rarely exhausted).

We also use *Pokémon* to define our baseline AI in terms of enemy action selection. This is a scripted AI, and the exact action selection process is not officially released by Nintendo, but the main decision criteria are reasonably well understood by player communities [20]. An enemy thus

- has an initial 70% probability of targeting a random player with a sleep or paralysis action, or
- uses a healing move (if available) on the ally with lowest health if it is below 25% of the maximum, or
- attacks the player with lowest health.

SLEEP

The inclusion of sleep as a combat action requires an AI implement two major decision tasks, first to decide which action to take (sleep or attack), and second to find the best target for the action. We assume here that a slept character is incapacitated for a fixed number of rounds > 1, symbolically represented as *SLEEP_DURATION,* that only one character in the player team can cast sleep, and the number of sleep casts is limited (we use a limit of 10, the same as in *Pokémon*).

Below we first analyze sleep to determine a solid basis for computing both benefit and lost opportunity cost, and then use this to offer an improved heuristic strategy for making the two main combat decisions mentioned above. We then undertake experiments to examine and compare our design in real combat scenarios inspired by the *Pokémon* games.

Cost and Benefit

Fundamentally, use of a sleep represents a cost-benefit trade-off. The primary benefit is of course the fact that an enemy is unable to attack, and so reduces the damage suffered by the player team (we do not consider any increase in the vulnerability of the slept enemy, a common trope which also constitutes a benefit). However, as casting sleep replaces an attack, it also represents a lost opportunity cost, reducing the rate at which damage is dealt to enemies, and thus increasing the overall duration of combat, and potentially the amount of damage received. We now formalize the cost and benefit, focusing on cost in terms of reduced damage dealt, and benefit in terms of reduced damage received.

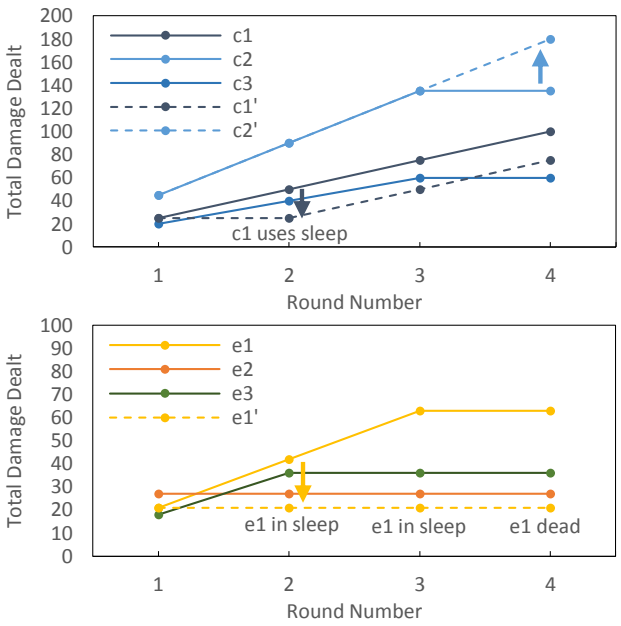

Fig. 1. Time *vs.* damage inflicted by player team (top), and taken by the player team (bottom).

Figure 1 illustrates the basic trade-offs. In the top graph we show 3 player team members, c_1, c_2, c_3 battling 3 enemies, e_1, e_2, e_3, plotting total damage dealt by the player team over time (round number). In the absence of sleep, each player does a fixed amount of damage to enemies, the curve flattening out once no viable enemies remain (in this we assume the enemies targeted by c_2 and c_3 die in rounds 2 and 3 respectively, the former event freeing c_2 to join in attacking e_3 on round 3, and leaving only c_1 fighting the remaining enemy). If player c_1 casts sleep on round 2, they do no actual damage on that round, resulting in a bend in their damage curve (marked by an arrow downward), shown as the difference between the solid line (no sleep used) and the dashed line (after sleep is used). This damage reduction is then compensated by player c_2, who must perform an additional attack in round 4 in order to help kill the last enemy.

The bottom graph in Figure 1 shows the benefit in terms of damage suffered by the player team. Whether sleep is used or not, enemy e_2 is killed in round 2 and enemy e_3 is killed in round 3. Without sleep enemy e_1 is actively attacking and inflicting damage until round 3 (solid line); slept in round 2, however, this part of damage done to the player team is eliminated (dashed line).

We can see from this example that the direct cost of using sleep is the damage loss by the casting player c_k foregoing an attack move (as well as the cost of using a sleep resource). Under our assumption that sleep casting requires one round, this is simply the caster's attack value, $c_k.a$ (and if sleep requires multiple rounds to cast this increases linearly, although we do not consider that here). The potential for *overkill*, and targeting choices, however, mean that this is actually an upper

bound, as the full attack power of the caster may not have been necessary in combat. Nevertheless, we use $c_k.a$ as a cost factor, and instead incorporate imprecision entirely into the benefit factor, as it has a larger impact.

Abstractly, benefit to the player team of using sleep is also simple, merely the product of the slept enemy's attack and SLEEP_DURATION. Depending on when sleep is cast and targeting choices, however, the slept enemy may die prematurely, during sleep, and thus it is again more correctly an upper bound on benefit, with the exact amount unknown at casting time, also dependent on targeting choices and overkill. As a lower bound, though, we know an enemy cannot be killed faster than if all players join in attacking that one enemy, giving a minimal survival time of a slept enemy of

$$r_{min} = 1 + \left\lceil \frac{\max(0, e_{slept}.h - \sum_{i \neq k} c_i.a)}{\sum_i c_i.a} \right\rceil, \quad (1)$$

where c_i ranges over players, and $e_{slept}.h$ is the health of the slept enemy. Note that this assumes the sleep caster is unable to join in attack on the round sleep is cast. A lower bound on benefit is then the product of the minimum of r_{min} and SLEEP_DURATION with the attack value of e_{slept}.

Decisions

A decision to attack or cast sleep depends on a (likely) positive cost/benefit trade-off. We have several cases to consider, giving us the following decision flow.

1) If $e_{slept}.a * \textit{SLEEP\_DURATION} < c_k.a$ then the largest possible benefit is lower than the cost, giving us a negative trade-off.
2) If $r_{min} = 1$ then e_{slept} has low health, and it may be possible for the player team to eliminate e_{slept} in one round. Sleep may still be effective, depending on targeting choices, but it has reduced value, and other options should be considered.
3) When $r_{min} \geq$ *SLEEP_DURATION* sleep is maximally effective, and thus well worth using.
4) If $r_{min} <$ *SLEEP_DURATION*, but $e_{slept}.a > c_k.a$, then we know that $r_{min} * e_{slept}.a > c_k.a$, and thus the trade-off is positive.
5) Finally, with $e_{slept}.a \leq c_k.a$ the benefit lies within the range $[e_{slept}.a * r_{min}, e_{slept}.a * \textit{SLEEP\_DURATION}]$. The trade-off here is ambiguous. However, as we require multi-round attention from the player team in order to eliminate e_{slept}, use of sleep can still be worthwhile.

If all enemies are in case 1 or 2, then sleep has no or quite limited value. A reasonable basis for choosing to use sleep is thus only when when an enemy can be found to fall in case 3, 4, or 5. We refer to this action choice heuristic as *smart sleep*.

Use of *smart sleep* must also, in general, select from multiple sleep candidates. Benefit is maximized by selecting an enemy with high attack, but the lower bound is improved by choosing enemies with high health, as they are not easily killed otherwise. Note that this is different from attack targeting heuristics, where low health and high attack has been shown preferable [3]. We will experiment with the actual targeting choices below.

Experiments

Experimental analysis allows us to evaluate the performance of *smart sleep* in practice, and to observe the impact of different targeting choices. For a more realistic game context, we use values from *Pokémon* (attack, health, and action sets), applying a typical player team of relatively average strength to a wide range of enemy teams, of varying sizes. The latter are controlled by an approximation of the *Pokémon* AI, as described earlier.

The player team consists of 3 pokémons ("Lapras" (#131), "Chandelure" (#609), "Garevoir" (#282)), selected to give us a team that includes agents with and without sleep capabilities (in these experiments healing is disallowed for both teams). Enemy team members are selected randomly from the entire *Pokémon* database, but balanced against the players by ensuring that a team's average attack and health do not vary more than ±20% from the player team. SLEEP_DURATION is set to 3 as the default in *Pokémon*. Player team size is fixed at 3, but a range of enemy team sizes between 3 and 6 is considered in order to simulate games of different difficulty, with the upper end of that range representing a game that is extremely challenging.

We considered 12 scenarios, independently varying 3 sleep strategies and 4 targeting heuristics:

Sleep Strategy	Targeting Heuristic
Smart sleep	Highest health
Random sleep	Lowest health
No sleep	Highest attack
	Lowest attack

No sleep only uses attack actions, and is a baseline that will show whether sleeping is useful at all, while *random sleep* chooses a sleep action 50% of the time (as long as a healthy target exists), and is intended to see whether the sleep heuristic is important. We simulate each combination and each enemy team size 1000 times. In each run, the choices of enemies are randomized (from Index 001 to Index 721 in the database [19]) to create a specific team size, which is then fixed for all strategy tests within that run to reduce noise in comparing strategies. We evaluate the results in terms of **WIN**, and averaged **HEALTH** scores.

Figure 2 shows the **HEALTH** results for all combinations (note that the y-axis scale changes for clearer visualization of differences among heuristics within each specific chart). In the 3-enemy scenario, we can see that *smart sleep* performs roughly the same as *random sleep*, and both better than *no sleep*. Players always move first in our combats, and so with equal team sizes, as long as some *sleep* is cast individual enemies tend to be killed very quickly, often in the first round. Even if the effect on health is similar, however, *random sleep* is less efficient than *smart sleep* in achieving this result: Figure 3 plots the average number of times sleep is used per

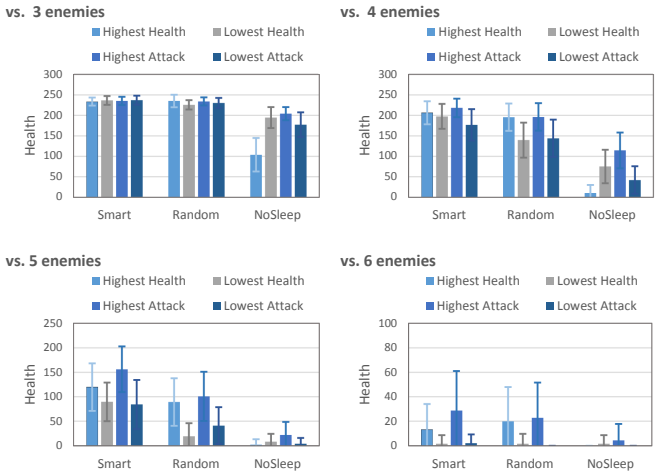

Fig. 2. Total remaining health of players; error bars show ±1 standard deviation

combat, and here it can be observed that *random sleep* uses significantly more sleep casts than *smart sleep*.

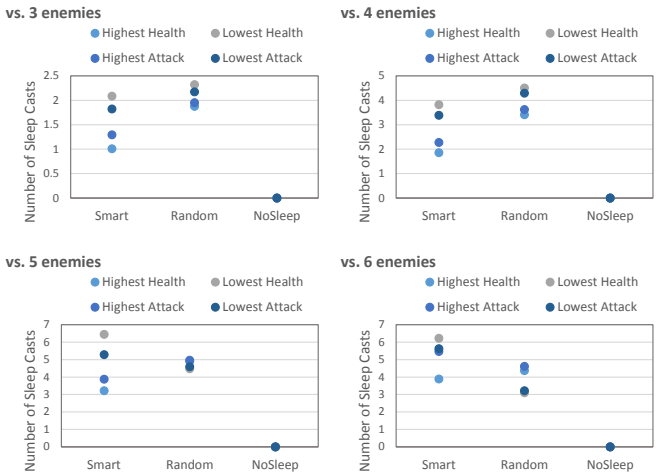

Fig. 3. Averaged total number of sleep casts per combat by players

As the combat becomes more difficult for players *smart sleep* starts to show increasing advantage over *random sleep*. With large enemy teams combat lasts longer, exposing cases 4 and 5 in the *smart sleep* heuristic, and more sleep actions are used accordingly, as seen in Figure 3. Sleep targeting strategies also begin to have an impact on the result in these tougher scenarios. *Highest Health* and *Highest Attack* begin to stand out as superior targeting tactics, confirming our earlier claim that an enemy with either high health or high attack power should be a better target to sleep. Our data shows the latter become preferable at high difficulty levels, although there is also large variance.

Figure 4 shows the **WIN** scores for all combinations. The trend here is similar to the **HEALTH** evaluation data. Surprisingly, however, in the 5 and 6 enemy scenarios, we notice that *random sleep* has very low chance of winning,

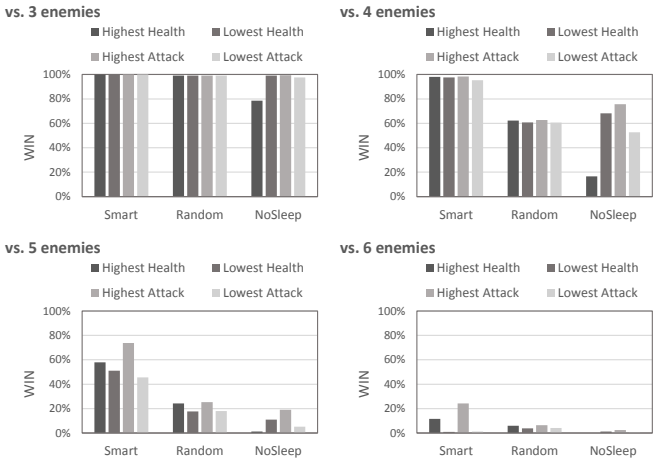

Fig. 4. Percentage win-rate for the player team

barely better than *no sleep*, and this despite the fact that its remaining health score (in winning situations) from Figure 2 does not differ that much from *smart sleep*. This suggests that *smart sleep* is better focused at helping win the combat than at ensuring maximal health, and also indicates the importance of not wasting sleep casts, especially when enemies are stronger.

HEAL

Healing actions have an effect similar to sleep in that they increase combat survival, although they do so by raising ally health rather than by reducing enemy attacks. We thus follow a similar analysis as for sleep, using a cost/benefit derivation to derive a choice heuristic, and then performing experiments to validate it. As with sleep we use a symbolic value for the most relevant parameter, using HEAL_AMOUNT to represent the amount of health restored by a heal action. In our case we set this to 50% of the target's maximum health, capped to avoid any overhealing. Again following *Pokémon*, heal casts are limited to at most 15 in a single combat.

Cost and Benefit

In our analysis of sleep, the benefit and the cost are represented in terms of damage decrease, from enemies and from the sleep caster respectively. For heal, the cost remains the same—by foregoing an attack action the player team loses an amount of damage equal to the healer's attack: $c_k.a$. Benefit, however, has no direct relation to the damage dealt by a player. In a trivial sense, the benefit is simply the health increase provided to the healing target—the HEAL_AMOUNT itself. Intuitively, however, healing is most useful when applied to a character who would otherwise be killed by the enemy, and this then relates to the amount of damage done to the enemies. The recovered portion of health potentially extends the healed agent's c_H's lifetime beyond a given round T_B by a number of rounds, Δt to $T_H = T_B + \Delta t$, and the damage dealt by c_H during this extra time is the gain for the player team, $\textit{benefit} = \Delta t * c_H.a$.

Calculating T_B and T_H, and thus Δt is hard because the values depend on the amount of attack directed at a given character, and thus targeting decisions. To ensure that using heal has advantage over using attack, we again find the minimum benefit and see if it is greater than the constant cost. Minimum benefit can be found be considering the smallest Δt, which is produced by the smallest possible T_H and largest possible T_B. The latter is maximized when few (or even no) enemies attack, while the former is minimized if all enemies switch attack to c_H immediately after heal is used. For simplicity, and reflecting the fact that in many real games enemies follow a scripted attacking routine that does not involve reprioritizing or changing targets while in combat, we assume attack strength applied to c_H is unchanging, and set it to the sum of attacks of all living enemies, $A = \sum_{e \in E} e.a$. This gives,

$$\textit{benefit} \approx \left(\left\lceil \frac{c_H.h'}{A} \right\rceil - \left\lceil \frac{c_H.h}{A} \right\rceil \right) * c_H.a \qquad (2)$$

where $c_H.h$ and $c_H.h'$ are the healths of c_H before and after healing respectively.

The timing of using heal also matters. Assuming any character loses health points healing will naturally improve the **HEALTH** result, but for **WIN** there is no point to using heal when it is unlikely to result in any increased damage dealt. We thus make another hypothesis that heal is used only when an ally is in danger—if one of the player team could potentially be killed within the next round, then saving it has a good chance of achieving a benefit, assuming healing can prevent the untimely death, which is guaranteed if health is raised above A, the sum of enemy attack values. Healing is thus most useful when $c_H.h \leq A \ \wedge \ c_H.h' > A$.

Decisions

Our decision heuristic directly follows the above reasoning. If healing is possible, we determine all players for whom $c_H.h \leq A \ \wedge \ c_H.h' > A$; these are allies of c_H (or c_H itself) which could benefit from healing. We then compare benefit as computed in formula 2 with the static cost $c_k.a$, filtering the candidate list to ones with a positive trade-off. This is our basic *smart heal* heuristic, and we combine this with a default targeting strategy of choosing a candidate with maximal benefit-cost difference.

Experiments

Experiments were again conducted using a setup similar to our sleep experiments. This time, however, we considered different enemy team sizes, of 2–5 and 8. Sizes 2–5 are meant to model situations in which the utility of heal ranges from ineffective to important to survival. For these we measure performance in terms of **HEALTH**. Enemy team sizes of 3–5 and 8 are measured with **DAMAGE**, as the upper bound of this set is effectively impossible to win, much like some boss fights. (We do not include enemy teams of 6 and 7 for space reasons, and as 8 is a better indication of the upper extreme.) We evaluate all sizes with **WIN**.

Different healing and targeting approaches are also compared. As well as our *smart heal* strategy, we consider *greedy heal*, healing an ally whose health falls below 50% of maximum, as a strategy similar to that used in many games. Again we include random choice, and a baseline of no healing. These strategies are multiplied by 2 different targeting heuristics, either choosing the candidate with maximal benefit as defined above, or simply a random target. This gives us 7 combinations (*no heal* does not have a targeting heuristic):

Heal Strategy	Targeting Heuristic
Smart heal	Smart
Greedy heal	Random
Random heal	
No heal	

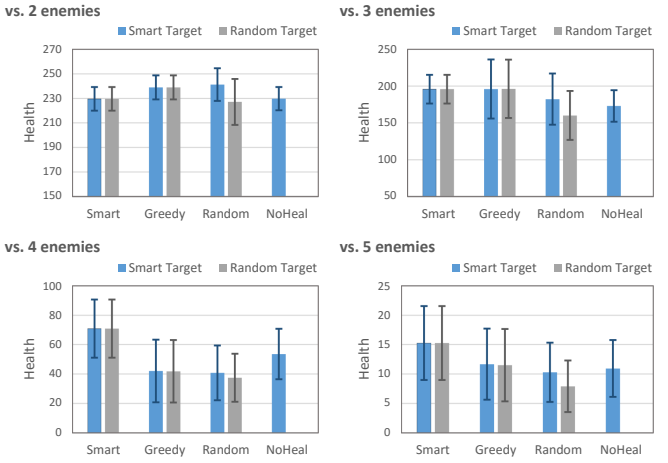

Fig. 5. Total remaining health of players against 2–5 enemies; error bars show ±1 standard deviation

Figure 5 shows that *smart heal* does have advantages over other strategies. An exception is for the size 2 enemy team; healing here is not required, and *smart heal* acts like *no heal*. The *greedy heal* and *random heal* strategies, however, achieve better **HEALTH** results, as they heal irrespective of whether it has a survival impact. As the combat becomes more difficult, and especially in the 4 and 5 enemy case, we can see not only that *smart heal* becomes the best strategy, but also that *greedy heal* and *random heal* start to perform even worse than *no heal*. With more enemies, it is possible for many enemies to target the same player and one heal might not be enough to allow c_H to survive. Evaluating the current combat situation and each agent's status becomes more crucial in deciding whether heal would be helpful, or would merely result in a lost attack opportunity.

Interestingly, the healing targeting strategy seems to have relatively little influence on the result. This is possibly caused by the enemy strategy of trying to focus on the same target as often implemented in modern games, so that most of the time we would have only one healing candidate with very low health. Smart targeting does improve *random heal*, however, as the improved candidate filtering partly compensates for the

random selection.

Fig. 6. Total damage dealt by players against 3–5 and 8 enemies; error bars show ±1 standard deviation

DAMAGE results are shown in Figure 6. Against 3 enemies, damage dealt is the same for all strategies, even though the remaining health is different—all enemies are dying, and so healing strategy does not affect total damage dealt. In more difficult combats against 4 and 5 enemies, *smart heal* shows advantages similar to the **HEALTH** evaluation. Notice that in these cases *no heal* also tends to do well. By not including any heal moves, attacks are maximized, and thus so is the total damage score. This comes, of course, at the cost of having a much lower remaining health, as seen in Figure 5. Once combat becomes greatly unbalanced, such as against 8 enemies, all strategies are again equalized. Healing here is no longer effective, as the large number of enemies can easily eliminate a player at full health in one round, leaving no healing candidates at all.

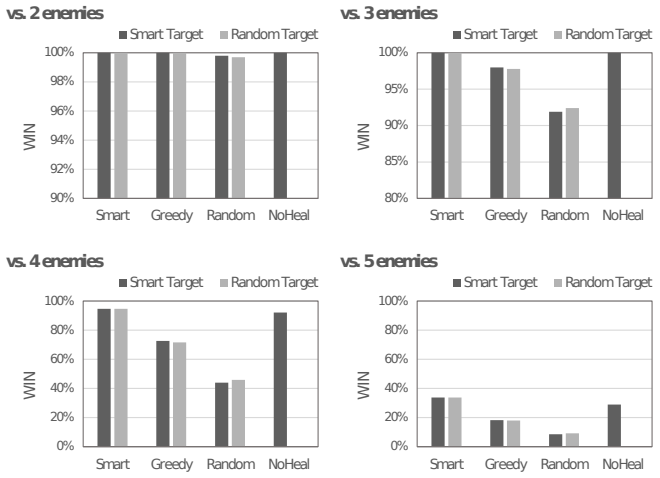

Fig. 7. Percentage win-rate for players

Finally, **WIN** results are shown for 2–5 enemies (8 is not shown as win rate is uniformly 0%) in Figure 7. In this and combined with the other data we can see that although *smart heal* is not necessarily better than *greedy heal* or *random heal* in terms of **HEALTH** for easier combats, the advantage of *smart heal* over other strategies gets larger and larger when combat becomes more challenging for companions, as long as winning is feasible, under all types of evaluations.

Sleep and Heal

Having used damage as a cost/benefit factor in both sleep and heal heuristics, combining our decisions for situations in which both actions are available is straightforward. For this we can test each heuristic independently. If candidates for both sleep and heal exist, then we choose the one with highest net benefit, breaking ties (arbitrarily) in favour of sleep. Targeting follows the strategies determined best for each case—highest attack for sleep, smart targeting for heal, and highest threat for the default attack action.

Experiments

The combination of sleep and heal can be fairly powerful, and so we evaluate our combined heuristic in more complex battle scenarios. We consider two main combat groupings; one with 3 players and enemy team sizes of 2–8, similar to sleep and heal experiments, and a second group of 10 players against 8–18 enemies. In these contexts agents in both teams are randomly selected, each is allowed all 3 actions (sleep, heal, attack), and we also remove the restriction on number of times an action can be used. Although this is now less representative of *Pokémon* itself, this gives us a richer and more balanced combat simulation, with less noise in terms of which agent can do what.

We evaluate our *combined smart* strategy along with *combined greedy* and *combined random* strategies. The greedy form is intended to be similar to the default *Pokémon* AI: if an ally has health below 25% then heal them, otherwise attempt to sleep the enemy with highest attack, otherwise attack. *Combined random* randomly chooses to sleep, heal, or attack with equal probability, and acts as a baseline to see whether any heuristic, even a greedy one, is truly necessary.

Figure 8 shows the **WIN** evaluation for both groupings (**HEALTH** is quite similar, and omitted for space reasons). In the 3-player grouping all strategies decline in effectiveness over a span of 3–4 enemy team sizes, but show a clear separation, with *combined greedy* and *combined smart* able to effectively compete against 1 and 2 larger enemy team sizes than *combined random*, respectively. *Combined smart*'s performance is quite good here, still succeeding at near a 90% rate against a 5-enemy team. In larger combat scenarios, however, this advantage is reduced, and while *combined smart* and *combined greedy* both improve over *combined random*, the separation is much smaller. Our *smart* strategies are based on lower and upper bound computations, and with greater numbers of enemies these ranges become larger with significant overlap, making decisions more arbitrary. We are not able to find best moves as consistently at large scales as we can in the small scale attrition games we have focused on. It is

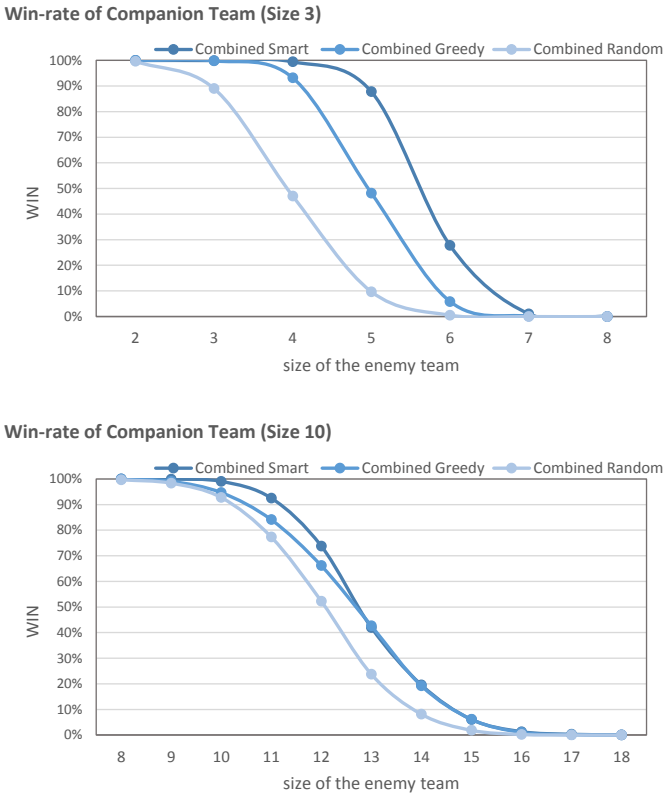

Fig. 8. Percentage win-rate for players with both sleep and heal

possible that less conservative estimation of benefit, perhaps including historical precedent or using probabilistic notions could restore the advantage, and is part of future work.

Conclusions & Future Work

Targeting problems in attribution games are computationally difficult. The addition of common but powerful abilities such as sleep and heal magnify this complexity, and although at their core they represent a conceptually simple trade-off between lost attack opportunities and the sleep/heal effects, the presence of unknown opponent (and companion) action choices and impact of varied parametrization in the skill use and effects make good decision-making non-obvious. Our derivation, heuristic design, and experimental evaluation show that effective and efficient decisions can be made based on relatively simple calculations, and can easily achieve better results than more traditional random, or hard-coded combat choices, for sleep and heal alone or in combination, without necessarily resorting to more expensive tree-search approaches. This makes our results usable in both turn-based and real-time, CPU-constrained contexts.

It may be possible to extend our approach to include other common combat abilities, such as defensive or offensive augmentation, as they have costs and benefits which can also be expressed in terms of damage received or dealt. More complex future work is aimed at incorporating combat geometry, which would let us consider the use of area-effects, where the target decision is more continuous, potentially affecting friendly as well as hostile units.

Acknowledgement

This work supported by the Natural Sciences and Engineering Research Council of Canada, Application ID #249902.

References

[1] T. Furtak and M. Buro, "On the complexity of two-player attrition games played on graphs," in *Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment Conference*. AAAI, 2010.

[2] S. Ontañón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A survey of real-time strategy game AI research and competition in StarCraft," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 5, no. 4, pp. 293–311, 2013.

[3] J. Tremblay, C. Dragert, and C. Verbrugge, "Target selection for AI companions in FPS games," in *Proceedings of the 9th International Conference on Foundations of Digital Games*, April 2014.

[4] F. I. Muhammad, "Graph searching implementation in game programming cases using BFS and DFS algorithms," Master's thesis, Sekolah Teknik Elektro dan Informatika, 2012.

[5] D. Churchill, A. Saffidine, and M. Buro, "Fast heuristic search for RTS game combat scenarios," in *Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. AAAI, 2012.

[6] M. Stanescu, N. Barriga, , and M. Buro, "Using Lanchester attrition laws for combat prediction in StarCraft," in *Proceedings of the Eleventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. AAAI, 2015.

[7] C. Browne, "Monte Carlo tree search," http://mcts.ai.

[8] B. Abramson, *Expected-Outcome Model of Two-Player Games*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1991.

[9] A. J. Champandard, "Monte-Carlo tree search in TOTAL WAR: ROME II's campaign AI," 2014, http://aigamedev.com/open/coverage/mcts-rome-ii/.

[10] C. D. Ward and P. I. Cowling, "Monte Carlo search applied to card selection in Magic: The Gathering," in *IEEE Symposium on Computational Intelligence and Games*. IEEE, 2009, pp. 9–16.

[11] A. Uriarte and S. Ontañón, "Automatic learning of combat models for RTS games," in *Proceedings of the Eleventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. AAAI, 2015.

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

[13] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Iowa State University, Tech. Rep., 1998.

[14] S. Morgan and M. S. Branicky, "Sampling-based planning for discrete spaces," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004.

[15] A. Bauer and Z. Popović, "RRT-based game level analysis, visualization, and visual refinement," in *Proceedings of the Eighth AAAI Conference on Artificial Intelligence for Interactive Digital Entertainment*. AAAI, 2012.

[16] J. Tremblay, A. Borodovski, and C. Verbrugge, "I can jump! Exploring search algorithms for simulating platformer players," in *Experimental AI in Games Workshop (EXAG 2014)*, October 2014.

[17] J. Tremblay, P. A. Torres, N. Rikovitch, and C. Verbrugge, "An exploration tool for predicting stealthy behaviour," in *Proceedings of the 2013 AIIDE Workshop on Artificial Intelligence in the Game Design Process*, 2013.

[18] S. Xu, "Improving companion AI in small-scale attrition games," Master's thesis, McGill University, Montréal, Canada, November 2015.

[19] "Pokémon essentials wiki," http://pokemonessentials.wikia.com.

[20] "Bulbapedia, the community driven Pokémon encyclopedia," http://bulbapedia.bulbagarden.net/wiki/Main_Page.