ChatGPT

Best Reward System for a Clash Royale Bot

Designing an effective reward system (i.e. **reward function**) is crucial for training a Clash Royale bot to consistently win. The reward scheme should reflect **both short-term achievements** (like dealing damage or taking enemy towers) and **long-term objectives** (ultimately winning the match). Below we outline an optimal reward structure, factoring in everything that can happen in a Clash Royale game, and how to distribute points accordingly.

1. Reward Components and Point Distribution

A comprehensive reward function should account for all key events in the game. Research on Clash Royale AI suggests using a combination of **dense rewards** (frequent feedback for incremental progress) and **sparse rewards** (major bonuses for achieving main goals) [1]. The main components include:

- **Tower Damage Reward:** *Yes – award points for damaging enemy towers.* Giving the bot a small positive reward for every bit of damage it inflicts on an opponent's tower provides continuous feedback. For example, one approach uses a **tower health difference** reward: as the enemy tower's health decreases (and/or your tower's health increases relative to the opponent's), the bot gains points [2]. This encourages the bot to make steady progress toward objectives. *(You can scale this reward to the proportion of damage dealt, e.g. a fraction of 1 point per percent of tower HP removed.)* Additionally, you can **penalize the bot when its own towers take damage** (negative reward for lost health) to encourage defensive play [2]. This way, the bot learns that keeping its towers healthy is as important as damaging the enemy's.

- **Tower Destruction Bonus:** *Give a large bonus for fully taking down a tower.* Destroying an enemy tower is a major objective, so it should yield a **significantly higher reward** than just dealing damage. For instance, a research-based reward design gives a substantial bonus whenever a tower is destroyed [3]. One example implementation is +20 points for taking an enemy Princess tower, and an even larger bonus (often 2× or 3× that amount) for destroying the main King's Tower [4]. In fact, because taking the King's Tower instantly wins the game, some reward functions weight it about three times as much as a normal tower [4]. This ensures the bot doesn't stop at just **damaging** towers – it learns to **finish them off**. Conversely, if the **bot's own tower is destroyed**, a hefty negative reward should be given (for example, –30 points) [5], since losing a tower is a major setback. By sharply differentiating partial damage vs. full destruction, the bot is motivated to push for complete objectives rather than just chipping damage.

- **Victory (Win) Reward:** *Reserve the highest reward for actually winning the match.* Ultimately, the bot's goal is to **consistently win**, so the reward for a victory should be very large relative to other rewards. This teaches the agent that **long-term success (winning)** outweighs any short-term gains. For example, one implemented Clash Royale RL agent gives **+300 points for a win** and **-200 for a loss** [6]. This huge gap makes it clear that winning is the paramount objective. You can similarly assign a big positive reward at the end of a match if the bot wins (and a big negative if it loses). By weighting the final outcome so heavily, the bot will prioritize strategies that lead to victory, rather than getting distracted by minor point gains. *In other words, no matter how many towers it damages, failing to win should feel very costly.* This aligns with your preference

for **long-term consistency** – the bot will learn to make sacrifices in the short run if needed, as long as it increases chances of winning eventually.

- **Tower Advantage (Crowns) Reward:** *Include a reward for tower/crown difference to encourage leading in objectives.* In Clash Royale, each tower destroyed is a "crown." You can reward the bot based on the **net crowns** it has relative to the opponent at any given time. A practical formula (used in one bot) was a logarithmic function that gives a small bonus for each tower lead and a penalty if behind, capping at about **+15 for a 3–0 tower lead** or **-15 for 0–3** [7]. This type of reward is more *coarse* than raw damage – it kicks in when a full tower falls – but it reinforces that having more towers down than the opponent is beneficial even before the final win. It's essentially an intermediate reward: larger than just a bit of damage, but smaller than the final win reward. By keeping this bonus relatively modest (e.g. +15 vs the +300 win reward in that example), it **helps the bot focus on gaining a tower advantage without overshadowing the importance of winning** [8].

- **Defense and Survival:** *Penalize losing towers or taking heavy damage.* As mentioned, negative rewards for your own tower health loss or tower destruction are important. Additionally, some designs give a small **constant "survival" reward each time step** the bot stays alive (+1 per second or per game tick) [9]. This can encourage the agent to avoid quick defeat (lasting longer yields more reward). However, be careful with such a **step reward** – if too large, the bot might learn to **stall** games needlessly just to accumulate points. If used, keep it small and ensure winning yields far more reward so the bot doesn't prefer dragging out a losing game. The general idea is to slightly reward **not dying**, which combined with the big win bonus, nudges the bot to both **survive and find a way to win**.

- **Resource Management (Elixir) Penalty:** *Incorporate a penalty for wasted elixir or inaction.* Clash Royale's elixir bar fills to 10; sitting at full elixir without playing cards is usually a mistake. To teach the bot efficient play, you can add a minor negative reward whenever the bot lets elixir overflow at 10 for too long [10]. For example, a small penalty (like -0.05 or -0.1 point per second of full elixir) was used in one research study [10]. This discourages the bot from idling or hesitating excessively. It will learn to **spend elixir proactively** rather than lose points for doing nothing. Similarly, some researchers even penalize using a card *inefficiently* – e.g. a small negative reward each time the bot plays a card [11] – to prevent reckless spam. You might not need a card-use penalty unless the bot tends to spam; the main goal is to ensure it doesn't sit at 10 elixir or play cards with no plan.

- **Enemy Troop Interactions (optional):** *Consider small rewards for combat outcomes.* The primary objective is tower damage and destruction, but defeating enemy troops is a necessary means to that end. You may give a **tiny reward for each enemy unit killed** to encourage the bot to make positive trades in battle. For instance, one approach awards +2 points for every enemy troop eliminated [12]. This helps the bot value taking out opponent's forces (since doing so usually protects your towers and clears the way to attack). If you include this, keep it much smaller than tower rewards so the bot doesn't fixate on hunting troops at the expense of pushing towers. The bot should see troop kills as helpful **toward** winning, not an end goal by themselves. (Likewise, you generally wouldn't reward spawning troops or minor actions themselves – let the bot figure out **how** to use cards to achieve the above rewards.)

2. Balancing Short-Term and Long-Term Rewards

It's important that these rewards are **balanced** so the bot learns the right priorities:

- **Emphasize Final Victory:** The largest reward by far should come from winning the match. This ensures long-term success is always the bot's primary goal. As noted, a win bonus on the order of 10–20× a single-tower reward is a good ratio [6] [8]. With such weighting, the agent won't sacrifice a win just to chase smaller points. For example, even if the bot could deal a bit more tower damage (small reward), it will do so **only if** it helps secure the win, because the win is worth dramatically more points. This aligns with **long-term consistency** – the bot is essentially trained that *"winning is everything"* in the end.

- **Dense Rewards for Learning Efficiency:** Giving points for intermediate actions (like tower damage, troop kills) is useful because it provides **dense feedback**. Without these, the bot would only get a reward at game end (win or loss), which makes learning slower and harder. By rewarding incremental progress, the bot gets hints about which actions are moving it toward victory. For instance, every bit of tower damage yields a small positive signal, so the agent starts recognizing that **attacking towers is good** [1]. This helps it learn strategies faster. **However,** make sure these intermediate rewards **do not overshadow the end goal**. They should be tuned such that achieving the ultimate objective (winning) yields the maximum payoff. If the bot ever finds a way to get points that doesn't correlate with winning (for example, hypothetically farming damage on a tower without trying to destroy it or win), you need to adjust the weights. In practice, designing the reward as described (small points for damage, big points for towers and victory) keeps the bot's incentives aligned with winning.

- **Avoid Perverse Incentives:** When crafting the reward function, consider all game scenarios to prevent the bot from "gaming" the system in unintended ways. For example, if you rewarded damage too heavily without a cap, a bot might prefer **spreading damage** across all towers to maximize damage points instead of focusing on actually destroying towers. To avoid this, the **tower destruction bonus** should outweigh the sum of damage rewards for that tower. That way, finishing off a tower is always more valuable than leaving multiple towers at half health. Similarly, the penalty for losing a tower should be significant enough that the bot doesn't ignore defense. A well-shaped reward function essentially makes **winning with as many of your towers intact as possible** the highest-scoring outcome, and losing with all towers destroyed the lowest-scoring outcome – with everything else falling appropriately in between.

- **Let the Bot Learn the Strategy:** With the above reward structure in place, the bot should be left to figure out *how* to maximize these rewards on its own through training. We are not hard-coding any particular strategy; instead, we're defining what "success" looks like in numeric terms. The bot will explore and discover that certain behaviors (e.g. deploying troops efficiently, defending at the right times, launching strong attacks when advantageous) lead to higher cumulative rewards. For instance, it will learn that **wasting elixir** incurs a penalty, so it will try to spend elixir in ways that contribute to tower damage or defense. It will learn that **overextending** and losing a tower hurts its score, so it will balance offense and defense. Because the reward function covers all important aspects of the game, the bot can develop a well-rounded playstyle by itself. This satisfies the idea that *"the bot should figure all of that out by itself"* – our job is just to ensure the reward incentives are correct.

3. Example Reward System Summary

Bringing it all together, here's an example of a **point distribution** that incorporates the above principles (you can tweak the exact values):

- **+1 point** for every *X* amount of damage dealt to enemy towers (dense reward for progress).
- **-1 point** (or similar) for every *X* damage your own towers suffer (to incentivize protecting your towers).
- **+50 points** for destroying an enemy Princess Tower, **+150 points** for destroying the King's Tower (game-ending tower).
- **-50 points** for each of your towers destroyed by the opponent.
- **+300 points** for winning the match (e.g. by having more towers at the end or taking the King Tower) [6].
- **-300 points** for losing the match.
- **(+15/-15 points** at game end based on net tower count difference, as a slight extra reward for 3–0 vs 0–3 outcomes, if not already accounted for) [8].
- **-0.1 point per second** of wasted max elixir (if elixir bar stays full without action) [10].
- **+2 points** for each enemy troop eliminated, **-2 points** if one of your deployed troops is wasted (dies without value) – this is optional and should be low impact [12].
  (Note: The numeric values above are illustrative; the key is their relative scale. Victory is valued highest, tower objectives next, and minor factors like damage or troops much lower.)

With such a reward system, the bot gets **points for both damaging a tower and taking it** – fulfilling your initial question with a **"yes"** (it gets incremental points for damage and a big payoff for actually taking the tower). The points are distributed in a way that **factors in everything important in Clash Royale**: dealing damage, destroying towers, defending your own towers, efficient use of elixir, and ultimately winning. This mixed reward strategy has been recommended in research and practice for training game-playing agents [1] [7], as it provides a balanced signal that guides the AI toward strong long-term performance.

4. Ensuring Consistent Wins

Finally, because the win/loss reward dominates, the bot will always seek to **maximize its chance of winning rather than just maximizing points**. In other words, the scoring system itself is set up so that **the highest score is always achieved by winning the game (preferably while preserving your towers and taking all of the opponent's)**. This aligns perfectly with the goal of long-term consistency – the bot will learn to make decisions that lead to **consistent victories** over time. Intermediate rewards (like damage or troop kills) serve to **shape the bot's behavior** and accelerate learning, but they are all in service of the end goal of winning the match [8].

By following this reward scheme, your Clash Royale bot will receive positive reinforcement for every useful action (offense or defense) and strong incentives to focus on the overarching objective of winning. It will gradually figure out the optimal strategies on its own, since the reward system correctly defines what "success" means in the game. In summary, **reward both damage and objectives achieved, weight final wins the most, and penalize setbacks** – this balanced approach will guide your bot to become a consistently winning player.

Sources: Reward design insights adapted from Clash Royale AI research and implementations [1] [4] [5] [7], which emphasize combining dense rewards (tower damage, etc.) with large sparse rewards for

tower destruction and victories. These ensure the agent learns effectively while always prioritizing winning the game.

---

1 3 [Literature Review] Playing Non-Embedded Card-Based Games with Reinforcement Learning
https://www.themoonlight.io/en/review/playing-non-embedded-card-based-games-with-reinforcement-learning

2 4 10 Playing Non-Embedded Card-Based Games with Reinforcement Learning
https://arxiv.org/html/2504.04783v1

5 11 12 Playing Card-Based RTS Games with Deep Reinforcement Learning
https://www.ijcai.org/proceedings/2019/0631.pdf

6 7 8 9 GitHub - Jaso1024/Real-Time-Strategy-RL-Clash-Royale: A Reinforcement Learning agent for Clash Royale created using Proximal Policy Optimization
https://github.com/Jaso1024/Real-Time-Strategy-RL-Clash-Royale