Computational Intelligence and Tower Defence Games

Phillipa Avery
Department of Computer Science and Engineering
University of Nevada, Reno, USA
Email: pippa@cse.unr.edu

Julian Togelius, Elvis Alistar,
and Robert Pieter van Leeuwen
Center for Computer Games Research
IT University of Copenhagen, Denmark
Email: {juto, elal, rpvl}@itu.dk

Abstract— **The aim of this paper is to introduce the use of Tower Defence (TD) games in Computational Intelligence (CI) research. We show how TD games can provide an important test-bed for the often under-represented casual games research area. Additionally, the use of CI in the TD games has the potential to create a more interesting, interactive and ongoing game experience for casual gamers. We present a definition of the current state and development of TD games, and include a classification of TD game components. We then describe some potential ways CI can be used to augment the TD experience. Finally, a prototype TD game based on experience-driven procedural content generation is presented.**

I. Introduction

Tower Defence (TD) games are a genre of strategy games focusing on resource allocation and unit (tower) placement. In it's simplest form, a TD game consists of a human player buying and organizing defensive towers that fire upon a set deployment of different types of offensive enemies (creeps). For each creep destroyed by the towers, the player earns resources corresponding to the difficulty of the creep. If enough of the creeps are destroyed, the player wins the round. If however enough creeps reach the end of the player's map, the creeps win. The tower deployment usually involves buying, placing and upgrading towers that fire automatically on the forces.

TD games have proved to be a challenging, addictive and fun way to pass the time. The simplicity of gameplay and the challenging tactics combined with the wide variety of games available in this genre has created a large population of casual TD gamers. It is this simplicity and availability that makes TD games a great test bed for Computational Intelligence (CI) research. Many of the popular video games used in CI game research are complex in nature, and are broken down into different components for the purpose of research. This often creates a less enjoyable game experience, and is not always a true depiction of the inherent game. Due to their popularity, TD games provide a large user-base with a simple game mechanism, and a range of interesting potential research possibilities. TD games are relatively easy to program being computationally and graphically simple. The TD genre is simple enough to implement completely for research purposes, while still providing enough of a challenge for user interest and complexity for test-bed purposes. Many TD games have tactical and strategic depth, meaning that there are real challenges to be solved and the results can have bearing on AI research in general. Further, there are clear possibilities to improve the gameplay of TD games using CI methods.

TD games can be real-time or turn-based in nature, and the allocation of towers allows resource allocation and strategic deployment. Additionally the choice and timing of 'creep' deployment also includes resource allocation and scheduling areas of research. The game can also be played in single player, multi-player (competitive and cooperative), and as described later in this paper, has the potential for a zero-sum two player game. All of these topics are popular problems to tackle using CI techniques, and the TD genre can provide a coherent test-bed. The popular use of online play for TD games also provides a potentially very large set of users to test with.

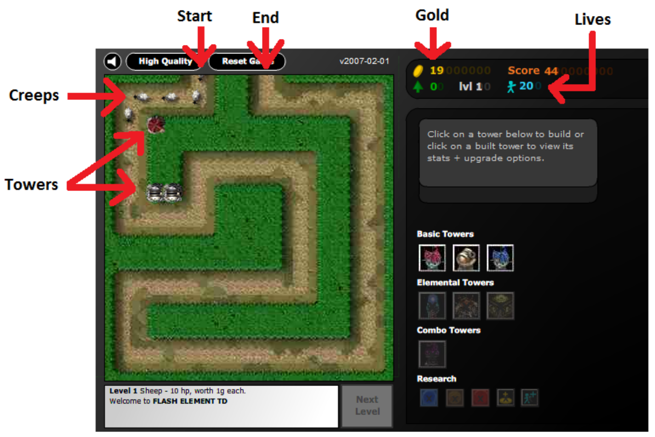
This paper defines the TD game for the purpose of future research, including all the sub-genres and off-shoots from popular TD games. The taxonomy of sub-genres provides increased complexity and interest, and can itself be a point of research potential with procedural content generation. We provide a brief history of the TD genre, including some of the most popular TD games. We discuss the possible reasons for why different games have become popular, and how future games created through research could cross into public popularity.

After providing the structure of TD games, we identify areas where improvements could be made to the game using CI techniques. We discuss a number of potential fields of interest, and how the use of TD as a test-bed could improve the field of expertise. Following this discussion, we describe a simple TD prototype that has been developed to demonstrate using CI techniques in the TD genre, and the results of some first experiments in evolving towers and creeps for this game.

A. An Example

As a simple example of a TD game, we provide a description of the Flash Element TD [1] that was inspired by the Element TD for Warcraft III (one of the first implementations of TD games). Figure 1 provides a screen shot of the game.

This game consists of three different tower types: one that has low fire power but targets both land and air creeps, and two that target either land or air creeps respectively. As you earn more Gold, you can buy more expensive but stronger towers, and upgrade your existing towers. In the figure, three towers have been placed on the map grid. In this version of



Click on a tower below to build or click on a built tower to view its stats + upgrade options.

Level 1 Sheep - 10 hp, worth 1g each.
Welcome to FLASH ELEMENT TD

Fig. 1. Example screen shot of the Flash Element TD game.

the game, towers can only be placed on the darker areas shown in green. The creeps follow the lighter brown path from the start to the end position. For each creep that reaches the end, a life is lost and the creep re-enters the map at the start position. The round (level) finishes when all creeps have been destroyed, and this game has a total of 31 levels with increasingly difficult creep deployment.

Due to the gentle learning curve and ease of playing in small chunks, this game genre is very popular with “casual” gamers; such players tend to play games that require little time investment and offer almost instant progress, and often do not think of themselves as “gamers” [2]. However, there is scope within the genre for “hardcore” play as well, as levels and game modes can be constructed that require significant strategic sophistication to win.

B. History of TD Games

The TD genre is a fairly recent one, gaining popularity particularly due to the online flash versions available for casual gamers. Arguably the first commercial TD game was the 1990 Atari arcade game *Rampant*. Due to its popularity, Rampant was subsequently released on other platforms, and went on to offer a multi-player version on the Playstation Network in 2007 [3], [4]. Although Rampart is not the typical TD game, it does involve defending a set of coastal castles by shooting attacking sea forces.

Following Rampant, a number of popular real-time and turn-based strategy games included TD style additions to the game. One of the most popular versions of these, and arguably the original true TD style game was the Tower Defense maps for the *Warcraft III* expansion *The Frozen Throne*. Using the map-making abilities of Starcraft and Warcraft III, players would create defensive maps similar to what we now know as TD games. When the expansion came out, a secret level held a TD game that demonstrated many of the TD elements seen in the classic version of the game [5]. The Warcraft III TD also allowed for multi-player game-play, where players compete to kill the most amount of creeps. This early link between Real-Time Strategy (RTS) and TD games highlights the similarities between the genres, which can be used to a researcher’s advantage.

TD games then grew in popularity with the use of Flash web games. There are a large number of popular flash TD games, and competition to gain players’ attention has created many variations on the original game. The following section describes some of these game elements, with reference to games that made them popular.

C. Elements of Tower Defence Games

The genre of TD games is varied, and there does not really exist a set of boundaries that define the genre. It is not our purpose to define a set of boundaries here, but instead to establish a set of features that TD games have displayed over the years. These features can be combined to make up the elements of a TD game. We ostentatiously define the genre by categorizing the game elements that the majority of TD games utilize.

1) Terrain: The TD map forms the constraints on how the player can allocate towers. In the classic TD game, the map is dominated by a linear path for creeps with surrounding area for building towers (as shown in Figure 1). This path can remain unchanged, or more commonly different levels can display more challenging paths. The map can further restrict the allocation by only allowing towers to be built in specific locations, such as specific terrain. One example of this is GemCraft, where players buy gems that can be placed in towers. A map begins with some towers created, and players can then build additional towers in permitting locations. Restricting usable terrain allows the creation of more difficult strategic deployment on an otherwise relatively simple game map.

One of the most popular Flash TD games is the *Bloons TD* games by Kaiparasoft. Bloons TD uses simple map design, with different maps categorized into easy, intermediate, advanced and expert. The map design has areas of high grouping impact, where towers can reach creeps (balloons in this case) for a longer time, and also areas where there is little impact. The easier maps are linear ones, with lots of high impact areas. The harder maps include branching paths, where creeps can randomly choose different paths to take. Figure 2 shows one of the maps from the Bloons TD 4 game, and is listed as an *expert* level map.

The Bloons example is typical of one genre of TD games. The game itself is simple, maps are easy to understand, and the graphics are pleasing yet uncomplicated. The complexity in the game comes from the unlocking of new and more interesting towers, and completing the levels with increasingly harder deployment of creeps.

Other popular games following this form of linear and branching paths include *Vector TD*, *GemCraft*, and many others. While many games include the path-type map design, with linear and branching paths, other games are more free-form. The classic example of this genre is the *Desktop TD*, where there is no designated map, and instead the tower placement forms the path design. Creeps at any instance attempt to take the shortest path towards the end of the

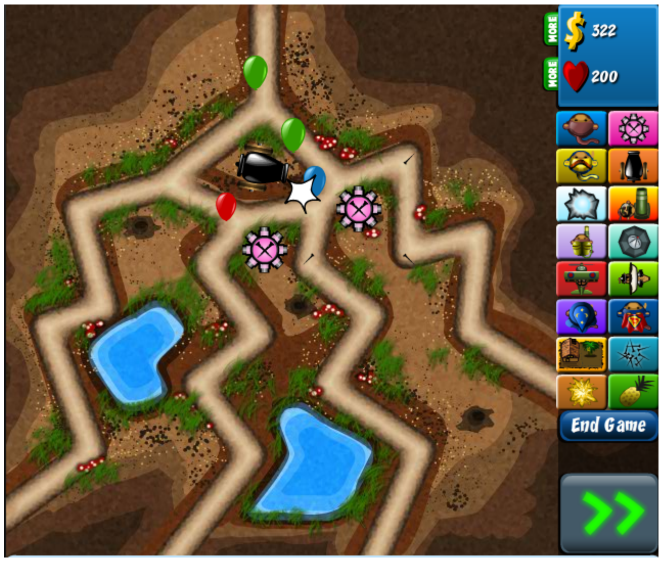


Fig. 2. Example screen shot of the Bloons TD game.

map, as constrained by the towers. Other games include maps where the placement of towers are limited (such as to only land in a land/sea terrain), and creeps can come from any direction. With this style the goal is not to defend the end position on a map, but to survive the round with your towers intact.

As a final example, a more recent desktop TD style game is the Plants vs. Zombies game. This game consists of parallel lanes where different towers (flowers) can be built to defend against the creeps (zombies). The creeps also follow the same lanes, and the areas for building towers is limited. In this version of TD, the creeps can destroy the towers to get to the end position. Once a single creep reaches the end, the game is over.

We can therefore distinguish between the following types of TD maps: *linear and branching paths*, *free-form path*, *survival* and *parallel lanes*. Naturally, hybrids between these are possible, as are entirely different forms of terrain.

2) Towers: The building of the towers requires strategic deployment and the allocation of resources. Most games allow towers that have different capabilities and costs. In the classic game, different towers target specific types of creeps (e.g. the land and air creeps as in the Element game discussed). Towers that can target multiple types of creeps are generally weaker. Towers also have different *firing ranges and speed*, *damage capability*, *attack formations*, and *effects*. These attributes can be improved through the purchase of more advanced towers, and through upgrading existing towers.

Many games specialize in providing a variety of different types of Towers. For example, the Bloons TD series discussed earlier has many towers that perform different functionality. The simple towers fire a single linear attack at a limited time interval. Other towers fire multiple attacks, attacks in different directions, attacks that damage multiple creeps at once, and 'affect' attacks that both damage and slow or stop the creeps. Later versions of the game also included towers that can locate hidden creeps, towers that gain increased resources and ones that can send creeps back to the starting position.

Most recent games either provide a large variety of towers such as Bloons and Plants vs. Zombies, or allow different functionality through changes to the towers. For example, the GemCraft TD game allows players to assign different gems to towers. The color of the gem determines the functionality of the tower, and the gems can be moved and combined to perform differently. The functionality includes such effects as slowing creeps, reducing the armour of the creeps, damaging multiple creeps at once (either in a splash or chain formation), damage over time attacks, damage multiplier, and generation of higher resources. Gems can then be combined for upgrade capabilities. Combining gems of the same colour provides towers with high specialized powers, while combining gems of different colours diminishes the power of the functionality, but allows a single tower to perform multiple functionalities. The gems can then be swapped and moved (with a loading penalty) among the existing towers, and new towers created to deploy the gems. This increases the adaptability of strategies during the real-time game play, and the combination of gems also adds increased interest.

Another game that provides combination of effects well is the Onslaught TD game. In this game, there are four main types of towers. These towers can be individually built and upgraded, but when they are placed in a specific formation they form combination attacks. For example, if a red tower fires on a creep first, followed by a blue tower, then this will generate a "Lazer Rocket" combo which is stronger than both towers. As the combo is created by whichever tower hits first, the range and where the tower is built both come into play. Onslaught 2 has a total of 20 combo attacks, each of which has a different graphic and affect.

Other games provide a 'hero' where you can choose a particular type of movable unit that has varying capabilities. This unit can normally be directed to a particular task, including directing them to attack a particular creep.

3) Creeps: The types of creeps used by the game are also directly influenced by the types of towers, and require a careful balance. Generally creeps consist of types matching the towers, for example air and ground creeps, or matching damage types such as ice, fire etc. They also have different abilities usually consisting of *speed*, *hitpoints* and *armour strength*. Slow creeps are normally harder to kill, while fast creeps have fewer hitpoints. The armour ability can be linked to a specific tower ability - such as a certain tower being able to lower or destroy a creep armour, thus making it easier to kill. The more difficult the creep is to kill (using any combination of the mentioned abilities), the more resources are generally obtained for killing it.

Most TD games also have a form of 'boss' creep, which are particularly hard to kill. These bosses tend to be deployed at the end of a round (when resources might be depleted), or on their own level. Generally if a boss reaches the end of the map, then the game is over.

4) Reward Systems: Most of the TD games are single player with a human playing against a static deployment of creeps. Once the particular deployment of creeps have been mastered, new challenges are needed (or the player moves onto another game). To increase the interest and longevity of TD games, different types of experience structures and achievement systems have been introduced. For example, the GemCraft series of games allows players to gain experience when a map is defeated, and can then use this experience to upgrade their overall abilities, such as increase all damage by a particular tower type. The TD game *Cursed Treasure: Don't Touch My Gems!* has a skill set that matches the three different types of towers available. As more skill points are earned and spent, different abilities are increased and unlocked. The experience structure encourages players to play levels more than once, to increase their abilities to a point where the next level is achievable. It also adds a new challenge mechanism to the game-play.

Other games and TD hosting websites have introduced achievement structures. These allow players to 'tick off' certain achievements gained through the game. These might include such things as number of creeps killed, number of gems combined, towers built etc. As well as challenging users, the achievements also encourage players to repeatedly play the game to achieve the harder achievement rankings.

5) Single or Multi-player: Most of the TD games are single player games, with a human player competing against a static set of creeps. Sometimes the creeps have some form of random deployment in timing or types, however this is restricted to maintain balance of the game. Multi-player versions involve players working in a cooperative way to beat the creeps, but competing to obtain the highest kill-rate.

D. Strategies

The strategies involved in TD games can be varied depending on the elements of the game. Different types of creeps require different strategies, and a well-designed game will be balanced to be challenging. Overall however, the strategies can be grouped into two main problem domains: resource allocation and unit placement. The resource allocation strategies usually involve choices between buying lots of cheaper towers and upgrades, or saving up for more expensive towers/upgrades. While it is common to buy cheap towers early on, one useful strategy is to buy minimal cheap towers and save for a powerful tower to place at a strategic location on the map. Other games that have different types of creeps often have creeps that come less regularly (such as air units). It can be tempting to forget to allocate resources to towers that target these units, and not have enough to counter waves of these creeps when they appear.

The placement of units is fully dependent on the map and the creeps being played against. While it is useful to have a strong front-line, and destroy creeps as they are deployed at the start position, fast creeps can potentially run straight past the front-line and additional towers further along the path are needed to stop this.

From the perspective of a game developer, it is necessary to balance all the elements chosen for the game with the creep deployment. For example, if there are towers with different abilities such as an area affect or slowing etc, the creeps should be deployed in such a manner to encourage use of these abilities. Having a range of creeps with different strength and speed is essential, and defining how to deploy these creeps to offer the most challenging experience for the user is an interesting balance problem.

II. Research Directions

We have described the basic TD game and the various elements that can make up a TD game, and now we discuss how the various research areas within computational intelligence are applicable. We begin with a discussion on the general usefulness of the TD genre to the academic community, and then offer specific areas that could benefit from a TD test-bed game.

A. Benefits of a TD research game

As mentioned previously, the TD game genre provides an opportunity for creating simple games that can be utilized for research purposes while being placed in the public domain. There are at least two good reasons to combine computational intelligence techniques with TD games: to find useful, fair and accessible benchmarks for comparing the performance of CI algorithms, and to find ways to improve the design and development of TD games, for example by enabling new game types or streamlining the development process. (In other words, the same motivations exist here as for research in CI and games in general.)

Regardless of the motivation, CI techniques could be applied to TD games in a number of different roles. Below, we attempt to survey the various roles CI could play in a TD game, pointing out related research that has been done on games from other genres.

B. Map Generation

The map is one of the core features of a TD game, and one of the main drivers of challenge and sources of differentiation between levels. One of the most common reasons a player stops playing a TD game is that there are no more levels to discover. In light of this, automatic map generation would be a major advantage for a TD game. Map generation could be done both offline, when the game is developed, or online, on the individual player's computer in response to the playing style and preferences of that player.

Map generation could be done in more or less simplistic fashions, for example through starting with a straight path and changing direction arbitrarily, or randomly placing objects on an empty surface and find the shortest path using a pathfinding algorithm. However, such methods are not very *controllable*, meaning that there is no good way of ensuring that the generated map is of the right difficulty, that it promotes the right kinds of towers and strategies, or even that it is playable.

A CI-based alternative is the *search-based* paradigm, where a stochastic optimization algorithm such as an evolutionary algorithm is used to search a content space for game content that maximizes certain criteria [6]. In case of a tower defence game, one would start with finding a representation for the map, and an operationalization of some desirable criteria for that map. These criteria could be that the map should have the right level of challenge, that it should promote combinations between certain towers, force a particular strategy etc. Once these criteria are formalized as a fitness function, maps can be evolved that satisfy them as well as possible given the game and map representation.

The search-based approach has previously been used to evolve maps for the real-time strategy game *StarCraft* [7]. In this experiment, the bases and resources on the maps were represented directly as (x, y) positions, whereas the rock formations that serve to separate parts of the map from each other were represented indirectly as parameters for a turtle-like (strongly defensive) mechanism. Several fitness functions, having to do with the fairness of the maps and their suitability for using advanced strategies (e.g. the presence of choke points) were defined, and combined using a multi-objective evolutionary algorithm. A similar approach could conceivably work well for TD games, once the appropriate map representation and evaluation functions are defined.

C. AI for playing the game

The deployment of resources in Real-Time Strategy (RTS) games has been a hot research topic in the last few years, with research performed on different ways to deploy units to on a map. Miles, Avery and Louis [8], [9] co-evolve the strategic deployment of units on an RTS grid using Influence Maps. Other research such as that by Weber and Mateas [10] focus on methods to automate the build order (when to build specific types of units); Hagelbäck and Johansson use multi-agent techniques and potential fields for controlling the units in RTS games [11]. These problems are also relevant to TD games, where the pertinent questions are where to build what towers and when. The TD map is also usually represented in a grid format similar to RTS games, so successful techniques for TD games might also be applicable for RTS and similar turn-based games.

The resource allocation area is another interesting research topic. Research on using CI techniques for a turn-based resource allocation game has been done by Johnson et. al. [12] and Avery and Michalewicz [13]. As a large component of the TD strategy is through resource allocation, it would be interesting to see if similar techniques could be applied here.

Also, the strategies involved for playing TD games can be varied, and success can be dependent on how and when creeps are deployed. While there is often a 'winning' solution, where a given strategy will always beat a given set of creeps, a more dynamic deployment of creeps could create some interesting strategy development. This concept lends itself particularly well to coevolution, where both successful creep deployment and tower strategies could be discovered for human game play. This is discussed further in the next section.

D. Creep strategy

One possible change to the TD game that the CI community can make, is to make the deployment of creeps dynamic or even dependent on the player's strategy. When using a dynamic creep deployment, agents could be created to play both sides of the TD game. This would require a method of restricting the way creeps can be deployed. One possible method of this is to treat creeps in a similar way to towers, in that the creep "player" earns resources by outlasting the towers. The further a creep progresses along the map path, the more resources earnt. These resources can then be used to purchase more creeps, with higher costs for stronger, faster and tougher creeps. It is then up to the creep player to determine what creeps to buy and when to deploy them in order to survive the opposition player's tower placement.

By changing the game in this manner, it is also possible for human players to play both sides of the game, as either creeps or towers. This provides many interesting research possibilities, as well as a more challenging game for human players. Play could either be done in real time, or turn based.

E. Game element generation

In addition to automatic map generation, there also exists potential to change the dynamics of the game itself. As discussed in Section I-C, there are a number of different variations on the TD theme. Some of these are easily exchangeable, such as changing the attributes towers (and corresponding creeps) are allowed to have. Finding a way to automatically choose and then balance these attributes could create another method of continuing game-play for a TD game. Demonstrating the ability for CI techniques to find continual balanced and engaging gameplay would also show the usefulness of CI techniques at doing this for commercial game prospects. The generation of elements and game AI could also be combined to automatically adjust the difficulty of the game. This is discussed further in the next section.

F. Dynamic difficulty adjustment and Player Modelling

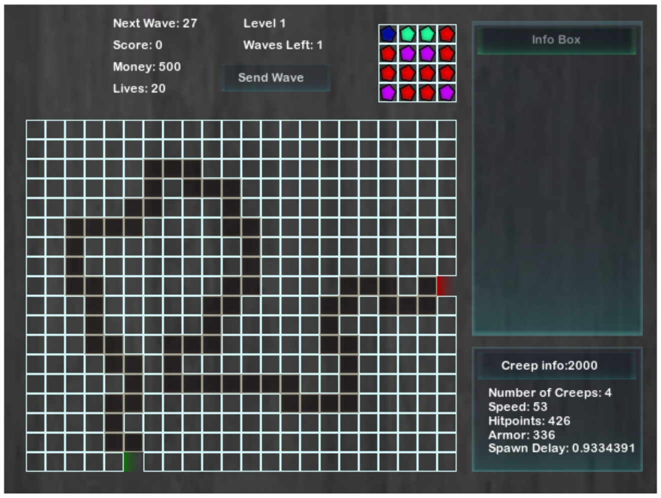
Game balancing, or *dynamic difficulty adjustment* (DDA), refers to the automatic adjustment of games to fit the skill level of the player [14]. This is motivated by the ever-widening demographic of game players, and ever-increasing costs of developing top-quality games. Simply put, a single game will need to appeal to a much larger range of different players and different playing styles now than it did back in the days when most game-players were teenage western males with a technology interest (and most games consequently were brutally punishing). The DDA answer to this challenge is to estimate the playing skill of the player and adapt the game in real-time so as to provide a reasonable level of challenge — not too hard, not too easy. Sometimes referred to as "rubber-banding", DDA has been criticized for removing the dramatic profile of games, and rewarding mediocre gameplay. Still, it is a commonly used technique, and how best to balance challenge is an ongoing research topic. It could be argued that the less obvious the adjustments are to the player, the better.

While difficulty adjustment could be implemented in a simplistic manner in a TD game, e.g. by lowering the health or speed of incoming creeps if the player is doing badly, more effective and less obvious methods could rely on map generation or creep strategy generation as discussed above. If the player is doing too well or too badly, next level's map could be evolved to be easier to finish, or the next creep wave could be evolved to be easier to beat given the existing weaponry. If the next level and wave will be dynamically generated regardless of the player's performance, this will make the adjustments hard to detect to the player. A similar suggestion for platform games is made in [15].

Another method of adjusting the difficulty is to create a model of the player's strategy, and develop a corresponding strategy specifically for that player. Work by Avery and Michalewicz [16] follows this method, and showed the potential for a fun and challenging game. Using this method in TD games could increase the enjoyment, when players know the game they are playing is tailored to their game playing experience. The grid nature of the TD game also gives potential for some interesting player modelling techniques, such as classification of tower placement and using th prediction techniques.

A further refinement that CI techniques make possible is to step away from a one-dimensional view of player performance, and instead see performance as having several components and adaptation to these as positioning the challenge profile in a multidimensional space. For example, a particular player might be good at placing the offensive towers for maximum effect, not very adept at using the supporting towers, make good investment decisions (i.e. when to buy towers and when to save money and get interest), be mediocre at selecting which gates to protect and have slow reactions to changing circumstances. These relative strengths and weaknesses can be measured and used to inform a fitness function, so that content can be evolved that plays the unique strengths and weaknesses of the player. This way, an appropriate overall different level can be maintained at the same time as the player is forced to explore new strategies and tower types.

In order to further personalize the game, the content generation can be made *experience-driven* [17]. This means adapting the game not only to the skills of players, but also to their preferences. For this, *player experience models* need to be created, mapping features of the game and gameplay to predicted player experience. Such models can be created through administering preference questionnaires after gameplay sessions, and mapping recorded gameplay features to the expressed questionnaires using neuroevolution [18]. This technique has previously been used to create neural network models that accurately classify levels in the classic *Super Mario Bros* platform game in several different affective dimensions (fun, frustration, interestingness etc.) for


Next Wave: 27
Score: 0
Money: 500
Lives: 20
Level 1
Waves Left: 1

Creep info:2000
Number of Creeps: 4
Speed: 53
Hitpoints: 426
Armor: 336
Spawn Delay: 0.9334391

Fig. 3. The Infinite TD game at the start of a level, before any towers have been placed and any creeps appeared. The 4×4 grid at the top of the screen is the tower repository, from which towers can be bought for placement on the map (or discarded).

individual players, allowing the creation of levels that elicit desired affects in particular players [19].

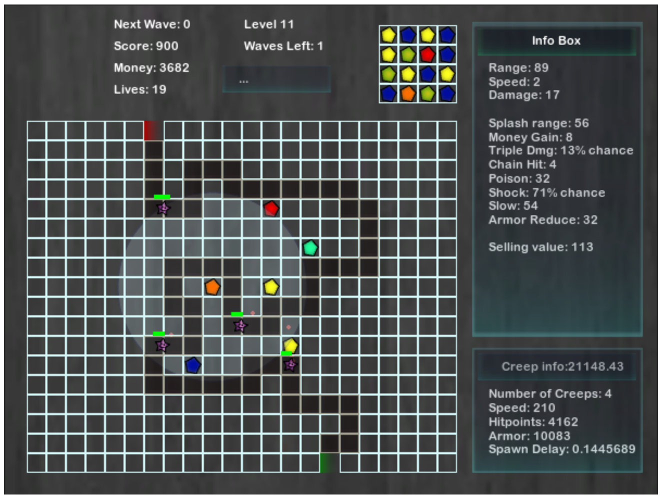
III. A Prototype CI-based TD Game: Infinite TD

Infinite TD is a prototype tower defence game aiming to demonstrate how CI and other adaptive techniques can be used to improve this genre of games. Instead of taking an existing TD game and modifying it, the decision was made to design the game from the ground up around adaptive mechanisms. This section describes the overall design and adaptive features at the current state of the game; the game is currently being developed, with a target of being released on major game platforms (e.g. the iPhone app store) late summer 2011.

A. Game Design

In some senses, Infinite TD is an utterly conventional tower defence game. The goal is to survive as many waves of creeps as possible. Every n waves, a new path is generated, and the difficulty of the new path is adjusted to counterbalance the player's performance. The creeps travel along the path from beginning to end without any possibilities for branching of diversion, and for each creep that reaches the end of the path, the player loses a life. To defend against the creeps, the player can purchase towers and place them strategically on the sides of the path. The price of a tower depends on its capabilities, which can vary along dimensions such as range, firing speed, poison effect etc., and money to purchase more towers is earned by killing creeps. Two screenshots of the game are shown in figures 3 and 4; the captions of those figures explain the elements of the game's user interface.

The non-conventional aspects of Infinite TD fall into three categories: map generation, creep evolution and tower evolution.


Next Wave: 0
Score: 900
Money: 3682
Lives: 19
Level 11
Waves Left: 1
...

Info Box

Range: 89
Speed: 2
Damage: 17

Splash range: 56
Money Gain: 8
Triple Dmg: 13% chance
Chain Hit: 4
Poison: 32
Shock: 71% chance
Slow: 54
Armor Reduce: 32

Selling value: 113

Creep info:21148.43

Number of Creeps: 4
Speed: 210
Hitpoints: 4162
Armor: 10083
Spawn Delay: 0.1445689

Fig. 4. The Infinite TD game during play of another level, in the middle of a creep wave. The purple stars inside the dark path are creeps, moving from the dark green path entrance towards the red path exit. Above each critter is a small green bar signifying its current health. The brightly coloured pentagons outside of the path are towers. One of the towers is currently selected, meaning that a dimly illuminated circle around the tower shows its range, and the info box to the right of the screen displays its statistics (range, speed, splash range etc).

B. Map Generation

An Infinite TD level is easier if the creep path is long and winding, and harder if the path is short and straight. One reason for this is that a long path gives the players more time to react to the creep wave, but the main reason is that a winding path effectively increases the range of towers, as they can be placed so that they cover larger parts of the path.

Each time the level of difficulty changes, a new map is generated based on the two parameters linearity and length, using a simple constructive approach. The algorithm selects one spawn point on a random edge of a map and a target point on another random edge of the map. The algorithm starts carving a path between these two points changing directions depending on the linearity value. If the algorithm gets stuck (e.g. path closes on itself), it backtracks to a previous point and changes the direction of carving. This process repeats until the carved path connects the spawn point and the target point and the path has the minimum length defined by the difficulty level.

C. Creep Evolution

At the beginning of a new wave, new creeps are evolved to be maximally effective against the previous strategy of the player, so as to force players to keep redefining and refining their strategies. Each creep wave is defined by the number of creeps, speed of the creeps, hit points, armor and spawn delay. When evolving a wave, the chromosome contains four genes in the range $0-1$; these are used to calculate the creep wave in combination with a number of $points$. The number of points act as a strength parameter for the creep wave, and can be used to adapt the difficulty of a wave without changing its composition. The wave is calculated from the genes as follows:

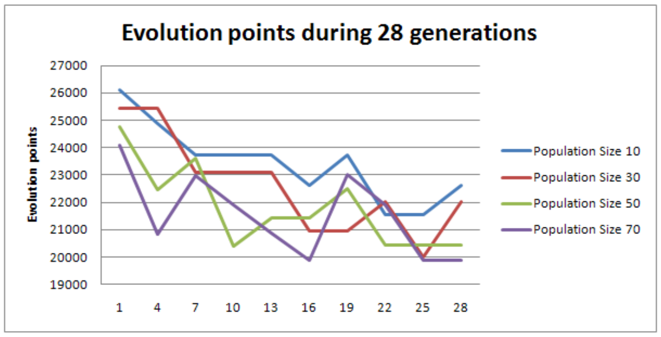


Fig. 5. Creep evolution. The “evolution points” value refers to the number of points that need to be spent before attaining a fitness of at least 1, meaning that at least one creep breaks through the player’s defences and reaches the end of the path. In this game, the creeps are tested against a complicated map with 30 towers.

- Amount of creeps in a wave: (gene 0) * 20 + 1
- Points per creep: (Total number of points) / (Amount of creeps + 2)
- Total weighing factor: (armor gene) + (speed gene) + (hitpoints gene)
- Speed: (speed gene)/(total weighing factor) * (Total number of points) * (Speed modifier)
- Armor: (armor gene)/(total weighing factor) * (points per creep) * (Armor modifier)
- Hitpoints: (hitpoints gene)/(total weighing factor) * (points per creep) * (hitpoints modifier)

The fitness calculation of a creep wave is simulation-based: the wave is played out against the player’s previous tower configuration (on the previous level) and the fitness value is calculated as the number of creeps that made it through to the end of the path, or the distance the creeps managed to travel along the path if no creeps made it to the end. This fitness is averaged over several playouts, as the outcome of a single game is slightly nondeterministic. It should be noted that thousands of waves can be simulated per second on a normal desktop computer, meaning that it is entirely feasible to evolve a new wave in just a few seconds.

Figure 5 shows the results of a series of experiments, where it was investigated how fast good creep wave configurations were evolved using a simple genetic algorithm and different population sizes. A good wave configuration is in this context one where a smaller number of points is needed to break through the defences and reach fitness 1 (one creep gets to the end of the path). The number of points needed is calculated through playing the wave starting with 2000 points, and if it doesn’t reach at least fitness 1 increasing the fitness with 5% cumulatively until that fitness is reached.

D. Interactive Tower Evolution

In order to keep the game challenging at all times we decided to give the players different towers that evolve according to their preferences. The towers are evolved using a genetic algorithm, and each individual uses chromosomes with 11 genes, 3 for the basic Range, Speed and Damage properties common and necessary for all towers, and 8 for the optional properties that add bonuses to the towers. The full list of the genes and their respective ranges is:

- Gene 0: Tower range: 50 - 200
- Gene 1: Shooting Speed: 1 - 4
- Gene 2: Splash Range: 0 - 80
- Gene 3: Damage: 3 - 100
- Gene 4: Slow: 0 - 80
- Gene 5: Resource Increase: 0 - 10
- Gene 6: Triple Damage: 0 - 80
- Gene 7: Armor Reduce: 0 - 80
- Gene 8: Poison: 0 - 50
- Gene 9: Shock: 10 - 80
- Gene 10: Chain Hit: 0 - 8

Each gene contains a float value between 0 and 1, which is translated into the specified range. For example, in order to find out the range of a tower that's the result of a value of 0.63 in Gene 0, we used the following formula: $geneValue * (maxRangeValue - minRangeValue) + minRangeValue$, which in this case means $0.63 \times (200 - 50) + 50$, resulting in a value of 144.5.

When the game starts the algorithm creates a population of 16 individuals. Two of these individuals have their genes initialized with completely random values, while the other 14 individuals have their genes initialized with values between 0 and 0.2. The initial range limitation is in order that the players don't get towers that are too powerful in the beginning of the game. Every time the player builds a tower, the population size increases with one and every time the player sells a tower, the population size gets decreased by one. The population size is always equal to the number of towers the player has on the screen at any time. When the player sells a tower or destroys one in the tower selection grid, a new tower needs to be generated. The new tower is generated as the first child from the one point crossover between two random individuals in the population. A Gaussian mutation is then applied to one of its genes. Every 15th tower that the game generates will be based on an individual with completely random values in its genes.

IV. Conclusion and Future Work

This paper has discussed the use of Tower Defence (TD) games in the CI field. We have given an overview of the game genre, and discussed the various elements that make up a TD game. We have also identified a number of interesting research areas where TD games would be particularly useful as a test-bed. Finally we introduce the first step in the creation of a game that investigates some of these research areas.

As shown, there are a wide and varied range of research possibilities for using the TD genre in CI research. The addition of CI techniques to the TD game could also create a very challenging and interesting game with commercially viable possibilities. In the future we hope to combine these two possibilities to create a game that the research community can experiment and learn from, and that the public can enjoy on an ongoing basis. The use of the game by the public will also make further data available for research.

Following on from the development of this game, we would like to create a competition to find even more interesting and exciting ways to extend the basic TD game using CI techniques. One way of doing this is to have a subjective competition, where the entries are placed in the public domain and the most popular (most played, or a voting system) entry wins. This could be in addition to an objective competition, where the most high scoring AI wins.

We hope that this paper will spark interest in the use of TD games in the CI field, and that future work will prove the importance of this simple yet strategically complex game.

References

[1] (2011, Jan.) Flash element td game. [Online]. Available: http://www.freewebarcade.com/game/flash-element-td/

[2] J. Juul, *A Casual Revolution: Reinventing Video Games and Their Players*. MIT Press, 2009.

[3] (2008, Aug.) The history of tower defense games. [Online]. Available: http://www.silvergames.com/weblog/history-of-tower-defense-games/

[4] L. Mitchell. (2008, June) Tower defense: Bringing the genre back. [Online]. Available: http://palgn.com.au/11898/tower-defense-bringing-the-genre-back/

[5] C. Tamas. (2010, Mar.) Understanding tower defense games. [Online]. Available: http://www.loopinsight.com/2010/03/30/understanding-tower-defense-games/

[6] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation," in *Proceedings of EvoApplications*, vol. 6024. Springer LNCS, 2010.

[7] J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelbäck, and G. N. Yannakakis, "Multiobjective exploration of the starcraft map space," in *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, 2010.

[8] C. Miles and S. J. Louis, "Co-evolving influence map tree based strategy game players," in *Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Games*. IEEE Press, 2007.

[9] P. M. Avery, S. J. Louis, and B. Avery, "Evolving coordinated spatial tactics for autonomous agents using influence maps," in *Proceedings of the 2009 IEEE Symposium on Computational Intelligence in Games*. IEEE Press, 2009.

[10] B. G. Weber and M. Mateas, "Case-based reasoning for build order in real-time strategy games," in *AIIDE*, 2009.

[11] J. Hagelbäck and S. Johansson, "A multiagent potential field-based bot for real-time strategy games," *International Journal of Computer Games Technology*, vol. 2009, 2009.

[12] R. Johnson, M. Melich, Z. Michalewicz, and M. Schmidt, "Coevolutionary optimization of fuzzy logic intelligence for strategic decision support," in *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 6. IEEE, 2005, pp. 682–694.

[13] P. Avery, Z. Michalewicz, and M. Schmidt, "A historical population in a coevolutionary system," in *IEEE Symposium on Computational Intelligence and Games*. IEEE, 2007, pp. 104 – 111.

[14] R. Hunicke and V. Chapman, "AI for dynamic difficulty adjustment in games," in *Proceedings of Artificial Intelligence and Interactive Digital Entertainment*, 2004.

[15] M. Jennings-Teats, G. Smith, and N. Wardrip-Fruin, "Polymorph: A model for dynamic level generation," in *Proceedings of Artificial Intelligence and Interactive Digital Entertainment*, 2010.

[16] P. M. Avery and Z. Michalewicz, "Adapting to human gamers using coevolution," in *Advances in Machine Learning II*, ser. Studies in Computational Intelligence, vol. 263. Springer Berlin / Heidelberg, 2010, pp. 75–100.

[17] G. N. Yannakakis and J. Togelius, "Experience-driven procedural content generation," *IEEE Transactions on Affective Computing*, vol. in press, 2011.

[18] G. N. Yannakakis, "How to Model and Augment Player Satisfaction: A Review," in *Proceedings of the 1st Workshop on Child, Computer and Interaction*. Chania, Crete: ACM Press, October 2008.

[19] C. Pedersen, J. Togelius, and G. N. Yannakakis, "Modeling Player Experience for Content Creation," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 1, pp. 54–67, 2010.