Turn Based Strategy Game

Pedram Amirkhalili

Supervised by: Prof. Sunil Arya

Introduction

Turn-Based Tactics (TBT) games are often plagued with long level design times. "Endless Tower", is a game that gets round this by randomly generated levels with the help of a genetic algorithm.

Player's must survive by outwitting an AI on varying maps, with nothing but their Army to help them.

Objectives:

1. Randomly generated levels
2. Genetic algorithm for generation
3. Basic gameplay mechanics

Figure A: The title screen

Motivation

TBT games repeatedly have two major issues, the first is that the level design process can be very lengthy. The second is without the correct and strenuous testing, levels can often be more difficult than they should be.

The lengthy design process can be reduced by randomly generating levels that fit a set of pre-defined criteria.

Fig. B shows a plot of Estimated Difficulty against Level Number, sometimes referred to as a difficulty curve. It shows a case in which the difficulty becomes wildly varied due to all control over the level being deferred to the random chance.

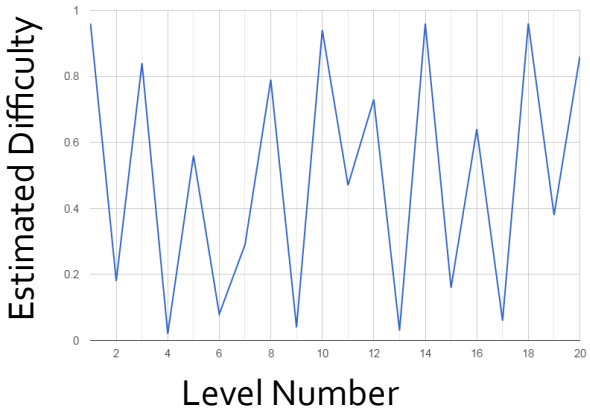

Figure B: Difficulty Curve that may occur due to randomly generating levels

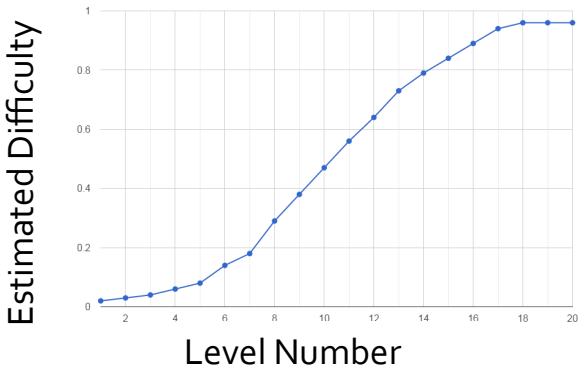

Figure C: The "Ideal" Difficulty Curve that most games try to emulate

The second major issue and the problem random generation causes (Fig. B), is resolved through the introduction of a genetic algorithm

The genetic algorithm evaluates the levels that are generated and breeds them in to find the "ideal" difficulty based on level number.

Random Generation

The levels for Endless Tower are generated in 3 parts:

1. Map
2. Armies
3. AI

The map is generated by placing different tile types into a large array, using varying methods dependent on tile type.

Then 2 armies, one for the player and one for the AI, composed of units from a pool of different types are created and placed across map to set up the battlefield

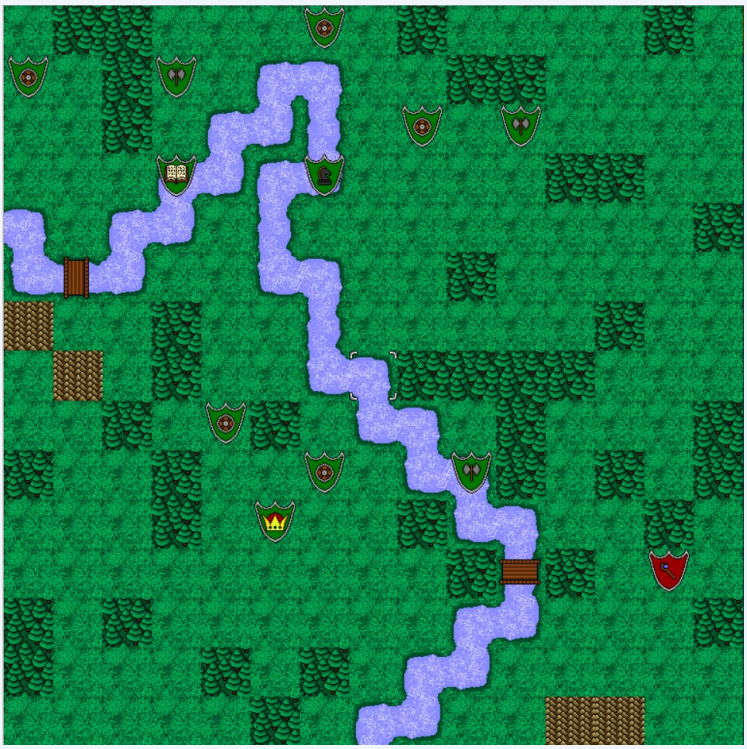

Figure D: An example of a randomly generated level.

The AI is simply selected to be either passive or aggressive, with a chance of having a priority targeting system.

Fitness Function

Once a sufficient number of levels have been generated they are passed to the genetic algorithm. Levels are evaluated by a Fitness Function on two fronts, the map and the strength of the player and AI armies.

The map was planned to be evaluated, as shown in Fig. E, on each "Choke Point" in it. The proximity to the player/AI determined it's score.

$$m(x) = \sum_{cP=0}^{n} ChokePointScore\ (\text{cP})$$

$$ChokePointScore = \text{closer to player} \begin{cases} \text{true} = \text{positive} \\ \text{false} = \text{negative} \end{cases}$$

$$cP \text{ is a individual chokePoint}$$

Figure E: The evaluation planned for maps, based off choke points

$$pU(x) = \sum_{u=0}^{n} UnitScore\ (\text{u})$$

$$UnitScore = \text{rating based on the level and value of the unit (level * value)}$$

$$u \text{ is a individual unit}$$

Figure F: The evaluation used for each unit in the player's army, this was done for the AI's army as well.

The armies of both player and AI are evaluated as shown in Fig. F, their respective scores are multiplied by a "Skill Level".

The new player score is subtracted from the AI's new score. This is then combined with the map component to get the overall score for the level.

The levels and are then bred through the genetic algorithm to find one that meets the current requirements as specified by Fig. C.

Conclusion

This project achieved the majority of it's objectives, and begun to solve two big issues in TBT games.

However there is still more that could have been done given more time:

- Completing map evaluation
- Background for the game world