Better Tower Defense – Milestone 3 Evaluation (November 25th)

Student: Jacob Dufault (jacobdufault@gmail.com)

Faculty sponsor: Professor Bernhard

Special Note

I ended up changing my priorities this milestone. Instead of completing the original milestone goals, I instead accomplished a large number of goals that needed to be completed sometime between some gameplay code being written and an extremely significant amount of gameplay code being written.

Progress Matrix (new)

Task	Completion
Implement serialization framework	100%
Implement level saving, loading, replays	100%
Replay/saved level versioning	100%
Separation of entity system from Unity	100%
Implement fully multithreaded entity system	100%
Extension platform	100%
Proper content editor support	100%

Progress Matrix (original)

Task	Completion
Implement resource system	0%
Implement power system	0%
Implement level types	0%

Task Summaries

Serialization Framework

I extensively evaluated four different frameworks used for serialization.

1. Unity's built-in serialization
2. Protobuf-net
3. NetSerializer
4. LitJSON

Each of the serialization frameworks had numerous problems. Unity could not support value types or generics, protobuf-net required explicit registration for inheritance hierarchies (problematic for code maintenance and an easy source of bugs), NetSerializer could not be used on mobile platforms due to code generation, and LitJSON was unusable because of an uncommented codebase (it needed additional features that it lacked, and the primary developer appears to have stepped down ~5 years ago).

In the end, I ended up writing a custom serialization framework that serializes to a language very similar to JSON, except with the extraneous characters removed. Performance has been good so far, and it supports a large variety of features. In is loosely based off of the structure of LitJSON.

Level Saving, Loading, and Replays

Saving and loading levels is a critical feature that needed to be implemented. It heavily uses the serialization framework. Due to the power of the already developed entity framework, all client code can have support for saving and loading with relative simplicity (they just need to serialize local state, which is easy given the flexibility of the serializer).

Replays are very similar to level saving and loading, except that each replay stores a list of input commands, which are then replayed as the game simulation moves along.

Replay/Saved Level Versioning

By utilizing a code loading mechanism for identifying gameplay code, replays and saved levels can be versioned so that any version of the game renderer can view them. For example, if a replay was made with version 0.5 of the game, then it will have associated gameplay DLLs that associate explicitly to version 0.5. When the replay is loaded, these DLLs are also loaded into the game and are used to provide the gameplay logic.

Separation of Entity System and Unity

In order to facilitate better debugging support (and in general tool support) and better code maintenance, the entity system now is completely independent of Unity. This means that all gameplay code is defined in an engine agnostic framework, which means that if a developer really wanted, Unity could easily be replaced with XNA/MonoGame (or similar) for rendering the game.

Fully Multithreaded Entity System

The entity system can now automatically run systems in parallel, providing a nice potential speedup as the number of entity systems increases. A significant amount of effort went into developing lock-free data structures data structures when executing systems to provide as close to linear speedup as possible.

Extension platform

As eluded to earlier, gameplay code is now loaded via DLL injection. This supports versioned saved states/replays, but it also allows for the game to be extended in a simple manner. The design of the entity system (automatic injection into systems) further supports the extension platform. Support for the extension platform has been possible as a nice side-effect of previous engine architecture design decisions.

Content Editor

Unity's built-in Inspector has a large number of seemingly arbitrary restrictions, such as complete lack of support for value types and generics. A similar inspector was written to support editing of all entities that are contained the entity system, even if they are generic or even if they contain value types. The improved inspector can be generalized to support more than just the entity system, but if that is done it is for a future milestone.

Interestingly enough, the new inspector allows for simpler code when defining complex data structures. For example, to create an effect (a la the dependent effect system) in the game requires only two classes (one for the settings and one for the runtime state), instead of the three before (settings, runtime state, Unity inspector wrapper).

Next Milestone

Title	Summary
Implement resource system	The resource system innovates in regards to not allowing some players to quickly gain a runaway lead in resource acquisition. Resources will be acquired by building a small base. For example, the player can build a *mine* to acquire resources, and a *blacksmith* to mine resources from the mine faster. Building a *library* can further increase productivity, and perhaps even allow for certain tower updates to be unlocked or allow for research towards powers.
Implement power system	The power system ties directly into the effect system. Powers are essentially effects that the player can trigger directly that impact a region of the map. The power system requires that resources be implemented, as triggering them will require resources.
Implement level types	A number of level types are going to be implemented, such as endless, waves, and attack. In essence, the level type describes the way that level is going to be played; for example, will it be cooperative with your friends or against your friends? More fundamentally, the level type most controls how spawning is controlled. For this reason, level types are going to closely integrate into the spawning systems.

Sponsor Feedback

Signature and Date: _____

Feedback:

Sponsor Evaluation – Better Tower Defense – Milestone 3

Jacob Dufault

Score (0-10):

Signature & Date: