Team Laser Combat

Capstone Postmortem

By Michael Henson

Introduction

With the capstone phase of the Team Laser Combat project coming to a close, a major milestone in the overall project has been reached. My stated goal of implementing alpha-level gameplay has nearly been accomplished, with the exception of a few bugs, some stability issues, and victory conditions which I accidentally neglected to include in my milestone requirements.

The aim of this postmortem, in addition to fulfilling a capstone requirement, is to document recent development efforts, step back and take a look at how these activities have gone, and plan the remainder of the project (as well as future projects) more effectively.

Pre-Capstone Preparations

Development of the game started long before my capstone phase began. In fact, the early design work started around the same time that I enrolled into DePaul University's Master of Science in Software Engineering program. This has been of great benefit to the project as I've been able to carefully consider several aspects of the game when using them as course assignments. For example, the object-oriented metrics scripts emerged from Professor Huang's class. Much of the technical design and planning was accomplished in Professor Streeter's class. The design patterns used for the character upgrade system were prototyped in Professor Jagadeesan's class. The state machine that drives the characters was adapted from Professor Burke's class. Several of the graphics were roughed out in Professor Schnepp's class. Numerous enhancements to the underlying game engine were inspired by three of Professor Keenan's classes. For so many reasons, this project is an appropriate climax of my educational experience at DePaul.

Defining Milestones

The capstone phase required three milestones, so I defined them with the following categorizations and requirements.

Milestone 1: Combat Initiation

The six requirements that comprised this milestone were centered around the functionality involved in enabling a unit to attack another unit (and in Milestone 3, a destructible container), including server-side validations, the user interface, the player's inventory, and related artificial intelligence.

Requirement ID	Description	Dependencies
1.1	The system shall perform a pre-validation of actions available to the player when it is that player's turn.	None.
1.2	The system shall construct a UI (the Actions Menu) which allows the selection of an available action based on validation results.	1.1
1.3	If the Attack action is available and is selected from the Actions Menu, the system shall display the directions in which an attack is possible and allow selection of one of these directions.	1.2
1.4	The system shall provide an Auto Attack option that players may select if they would rather have the game select their tokens for them automatically after issuing the Attack command.	1.3
1.5	After the player has selected a direction for attack, the system shall display the offensive combat tokens from the player's inventory and allow the selection of a token to spend IFF the Auto Attack option is turned off.	1.4
1.6	AI players shall be able to issue Attack commands that are subject to the same validation rules as the human players.	1.4

Milestone 2: Combat Resolution

The seventeen requirements of this milestone build from the foundation established by Milestone 1 and complete the functionality for combat interactions. This includes some extremely challenging elements -- the processing of all ten types of combat resources, complex coordination between the server and the clients, some of the most detailed user interface screens in the entire game, and more AI.

Requirement ID	Description	Dependencies
2.1	When a unit is attacked, the system shall notify the player controlling the unit by presenting a Defend UI which allows	1.5

	selection of a defensive combat token from the player's inventory.	
2.2.1	The combat evaluator shall process Combat Tokens for their basic attack and defense values.	2.1
2.2.2	The combat evaluator shall process the Vampire Power Token | If the attack does damage, the attacker heals one hit point per unit of damage dealt.	2.1
02/02/03	The combat evaluator shall process the Lucky Shot Power Token | If the attack does damage, the damage amount is doubled.	2.1
2.2.4	The combat evaluator shall process the Desperation Power Token | The attack value is normally 4, but if the attacker is the only remaining member of the team, then the attack value is 10.	2.1
2.2.5	The combat evaluator shall process the Blaze of Glory Power Token | If the attack does damage, the damage amount is doubled, but if the attack is blocked, the attacker takes 5 points of damage.	2.1
2.2.6	The combat evaluator shall process the Reflection Power Token | Any attack, regardless of power, is reflected back at the attacker. The defender takes no damage.	2.1
2.2.7	The combat evaluator shall process the Resolve Power Token | Defense value goes up as the character health goes down. DF = maxHP - currentHP	2.1
2.2.8	The combat evaluator shall process the Lucky Break Power Token | If the attack is blocked, any excess defense points are converted to health points for your unit.	2.1
2.2.9	The combat evaluator shall process the Shatter and Scatter Power Token | If the attack is blocked, each member of the attacking team takes two points of damage.	2.1
2.2.10	The combat evaluator shall process Grenade Tokens | The attack functions like a Combat Token but cannot be blocked, has no defense value, and does two points of damage to all characters or containers within a two-meter radius of the primary target.	2.1
2.3	The game server shall broadcast a summary of combat results to all connected players.	2.2
2.4	The system shall provide a Combat Results toggle that allows players to opt out of seeing the detailed combat summaries at the conclusion of each combat interaction.	2.3
2.5	The user interface shall display a combat summary screen when combat results are received from the game server IFF	2.3, 2.4

	the Combat Results option is turned on.	
2.6	The system shall provide an Auto Defense option that players may activate if they would rather have the game select their tokens for them automatically when their units are attacked.	2.1
2.7	The system shall display an event log on the HUD which records the actions taken by all units in the game.	2.3
2.8	AI players shall be able to defend against attacks subject to the same rules as the human players.	2.6

Milestone 3: The Game World Environment

This was the simplest milestone. The nineteen requirements really boiled down to two categories of functionality: bonus spaces on the game board and destructible containers.

Requirement ID	Description	Dependencies
3.1.1	The system shall place a **coin** bonus item in a random, unobstructed location on the game board at the start of each match.	None.
3.1.2	The system shall place a **token** bonus item in a random, unobstructed location on the game board at the start of each match.	None.
3.1.3	The system shall place a **snack** bonus item in a random, unobstructed location on the game board at the start of each match.	None.
3.1.4	The system shall place a **weapon upgrade** bonus item in a random, unobstructed location on the game board at the start of each match.	None.
3.1.5	The system shall place an **armor upgrade** bonus item in a random, unobstructed location on the game board at the start of each match.	None.
3.1.6	The system shall place a **boots upgrade** bonus item in a random, unobstructed location on the game board at the start of each match.	None.
3.2.1	When a unit travels onto a space occupied by a **coin**, the system shall add one coin to that unit's team's budget.	3.1.1
3.2.2	When a unit travels onto a space occupied by a **token**, the system shall draw one random token from the team's token source and add it to the team's inventory.	3.1.2
3.2.3	When a unit travels onto a space occupied by a **snack**, the system shall add one point to the unit's health up to that unit's maximum.	3.1.3

3.2.4	When a unit travels onto a space occupied by a **weapon upgrade**, the system shall apply a free upgrade to that unit's weapon.	3.1.4
3.2.5	When a unit travels onto a space occupied by a **armor upgrade**, the system shall apply a free upgrade to that unit's armor.	3.1.5
3.2.6	When a unit travels onto a space occupied by a **boots upgrade**, the system shall apply a free upgrade to that unit's movement stats.	3.1.6
3.3	The system shall remove a bonus item from the board after its effect has been applied.	3.2
3.4	The system shall place one invisible hazard in a random, unobstructed location on the game board at the start of each match.	None.
3.5	When a unit travels into the space occupied by the random hazard, the system shall deduct one hit point from the unit.	3.2
3.6	When a unit is damaged by a random hazard, the user interface shall display a notification to all players indicating which unit was damaged but not revealing which space contains the hazard.	3.5
3.7	While loading the data from a map file, the system shall recognize identifiers for destructible containers and instantiate them onto the board accordingly.	None.
3.8	The attack validator shall recognize destructible containers as valid targets to be attacked.	1.1
3.9	When a container is attacked, it is removed from the board and replaced by a random bonus item: a coin, a token, a snack, a weapon upgrade, an armor upgrade, or a boot upgrade.	3.1, 3.8

Scheduling

The metrics selected for the project were as simple as can be: number of requirements and length of time. Milestones 1 and 3 were completed within their timeframes. Milestone 2, however, overflowed its bounds and still has defects that have not been corrected.

Milestone	Number of Requirements	Planned Duration	Actual Duration
1	6	21 days	20 days
2	17	28 days	42 days
3	19	14 days	6 days
TOTAL	**42**	**63 days**	**66 days**

Testing

My primary method for testing was somewhat informal. I would set up a specific scenario in the game, and then systematically test each requirement of interest. Of the 42 total requirements, 39 now have correct functionality and 3 are defective. In addition to the requirements defined for the capstone phase, there are numerous defects due to the multithreaded client/server architecture, but I am treating those separately since they are outside the scope of the capstone phase.

Lessons Learned

What Went Right

- I think the best thing working in my favor was that I had done a thorough design and had a clear understanding of all of the project's requirements before my capstone began.
- Defining clear, unambiguous requirements statements for each milestone helped to keep the effort focused and give me a gauge for verifying and validating the requirements.
- Implementing the observer pattern in the network communication modules was brilliant. The original implementation did not do it this way, but it is a far superior design.
- I have learned something important about the way I work: Working on these focused milestones with specific requirement statements has given me the clearest, most measurable progress of any project I've ever done. I have learned through this experience that randomly piecing a game together in a strictly linear fashion is not necessarily the best approach. The best approach is to design the game software with solid milestones like I did here, paying special attention to dependencies and planning the order of the milestones accordingly.

What Went Wrong

- The program should have been data-driven from the start. The functional inconsistency between the client, the server, and the AI multiplied the testing effort.
- I should have used the remote proxy design pattern for all clients, including the AI, rather than requiring single-player mode to rely on network connectivity. The added complexity was a large factor in why the milestones took as long as they did, and why Milestone 2 never reached full completion. The worst part is that my original technical design specified this, but the detail was lost when implementation began.
- It was a bad idea in this case to have a single facade for all game data. This works great for single-player games, but for a networked multiplayer game such as this, the common global data should have been held separately from the player-specific data.
- Dynamic and finely-placed objects should not be part of the game map's data structure. The engine's map rendering algorithm uses coarse object placement as a matter of necessity, which makes it impossible for an object to lie randomly on the ground, or for a character to walk smoothly from one map cell to another. Real-time physics is completely out of the question.
- Testing coverage should be consistent throughout a project. In my projects, thorough testing tends to happen early on, but as deadlines approach, it often gets sacrificed as it did in Milestone 2. If one must cut corners, testing is not a good place to do so.
- Model-View-Controller principles are *always* relevant, even beyond the basics of the Single Responsibility Principle. I am beginning to think that the high-level architecture should always begin this way and then evolve to fit the individual needs of each project. At the very least, this is worth further consideration.

Summary

Without a doubt, the Team Laser Combat project has been the greatest learning experience of any project I have ever done. I have done a lot of things right and a lot of things wrong, but every aspect of the project has come with valuable lessons to guide me on future projects. I am excited about this game, but I am more excited about the prospect of doing bigger and better games in the future, guided by the lessons learned here.

I am committed to finishing the game in at least a single-player version, but I have not given up hope of making the multiplayer work just as well.