The Zeven Trials

The Beta Boyz Of C-Block

Team Chronos

Ross Visser, Luke Leontowich & Shannon Abeda

Due: March 18, 2020

Table Of Contents

INTRODUCTION

Dr. Anvik assigned us a term project where we design and build a text-based rpg game using C++. Our team successfully was able to meet the required deadline and complete the required project. We created a rpg game with seven different levels and seven unique puzzles where the player must pass each stage in order to win.
Our game was able to compile successfully and run with no memory leaks, static or style errors.

PROJECT MANAGEMENT

TEAM ORGANIZATION

Shannon Abeda: Tester
Ross Visser: Team Lead, Developer & Main Designer
Luke Leontowich: Developer & Quality Assurance Lead

Team Roles

Team Member	Role for current Phase
Ross	Phase Lead
Luke	Implementation Lead
Shannon	Testing Lead

Minutes from meetings:

Tues February 18:
Group met in computer lab D and begin initial coding of project.
Worked on setting up include, test and src files.
Got code blocks configured and made sure everyone had the same version.

Thursday February 20:
Group met in computer lab D.
Implementation of src.
Setting up testing for test driven development
Ross brought coffee for the team!

March 5:
Group met in computer lab D
Exceptions class created
Makefile and Doxygen started

March 7:
Worked together on minor errors

March 9:
Discussed what is left to be done and what each team member is going to work on

March 15:
Talked about deleting character class since enemy and player class do two different things and thus don't need inheritance from a character class

March 18:
Met to do last minute changes.

RISK MANAGEMENT

To manage the workload that was given to us, our group decided to start on our project during the reading break. We were able to complete the majority of our implementation and had a working game completed a week before the deadline.

We had anticipated the amount of work required and scheduled accordingly. We all committed to 3 hours during the reading break per day and whatever time we had during the last two weeks of implementation.

As a result, we maintain a strict timeline which alleviated stress and potential circumstances that might sidetrack us. Before the CS labs and school shutdown due to Coronavirus, our project was fully complete which eliminated the need to do major work remotely.

DEVELOPMENT PROCESS

CODE REVIEW PROCESS

- Team members proposed changes went through the phase lead for each phase, addressing how they think their idea or change will improve the project
- The phase lead decided if the idea would be beneficial to the game or add more confusion and was implemented, adjusted to scrapped accordingly.
- All large changes to the game were addressed as a team and given the opportunity to address concerns or issues with the new idea.

COMMUNICATION TOOLS

Our communication tools were primarily an iPhone iMessage group chat as well as communication before and after the 2720 class times everyday. Any updates to the repo were also communicated via the group chat every time a push occured.

CHANGE MANAGEMENT

- The QA lead was responsible for all bug reports and assignment of who fixed the bug. The individual that was responsible for the code portion that created the bugs was assigned to fix it if it was a large issue, but any small issues were handled by the QA.
- Any bugs that were caused by multiple people working on the same code collaborated to come up with an optimal solution to the problem.
- The individual that handled the bug made a comment on the code stating what was done to fix the issue.

SOFTWARE DESIGN

DESIGN

The game was designed to have the Game class be responsible for all the work done in the game and have every other class be a helper to running the game, the Main.cpp creates a Game type item, calls play and ends. The main design is a single function (play) that will start the player in the game and run through the battles and puzzles until they get to the end of the game or die. All of the weapons in the game are accessed using the GameItems class and return pointers to locations in a vector of all the players items. The Game classes use Player and Enemy pointers to store them, so that there aren't any possible memory leaks passing ADT’s into functions.

DESIGN RATIONALE

Code follows SOLID principles + DRY

S - Single Responsibility

Each class exists to serve a single purpose

- Player (base class)
    - Stores all the information for the player over the course of the game
- Enemy (base class)
    - Only exists to fight player in battles
- Items (base class)
    - Shield (sub class)
        - Shield value used for blocking damage in battle
    - Armour (sub class)
        - Constant value for blocking damage in battle
    - Weapon (sub class)
        - Stores the damage value of the weapon to be manipulated during the battle

O - Open/Closed Principle

Each of our classes were designed to easily accommodate any new functionality we see fit. This was done by creating simple functions that can be manipulated to help implement any new functions.

L - Liskov Substitution Principle

The substitution principle works in reference to the Items class and its subclasses (weapon, shield, armour). Where each class only has a name and int value. Originally we had a Character base class that creates Enemy and Player subclasses, but after looking at the implementation we realized the two classes were too different to be considered subclasses so we made each of them their own classes.

I - Interface Segregation Principle

Every function and class in the game is designed to the interface of the Game class. All the classes that are not the Game class are used to assist the Game class.

D - Dependency Inversion Principle

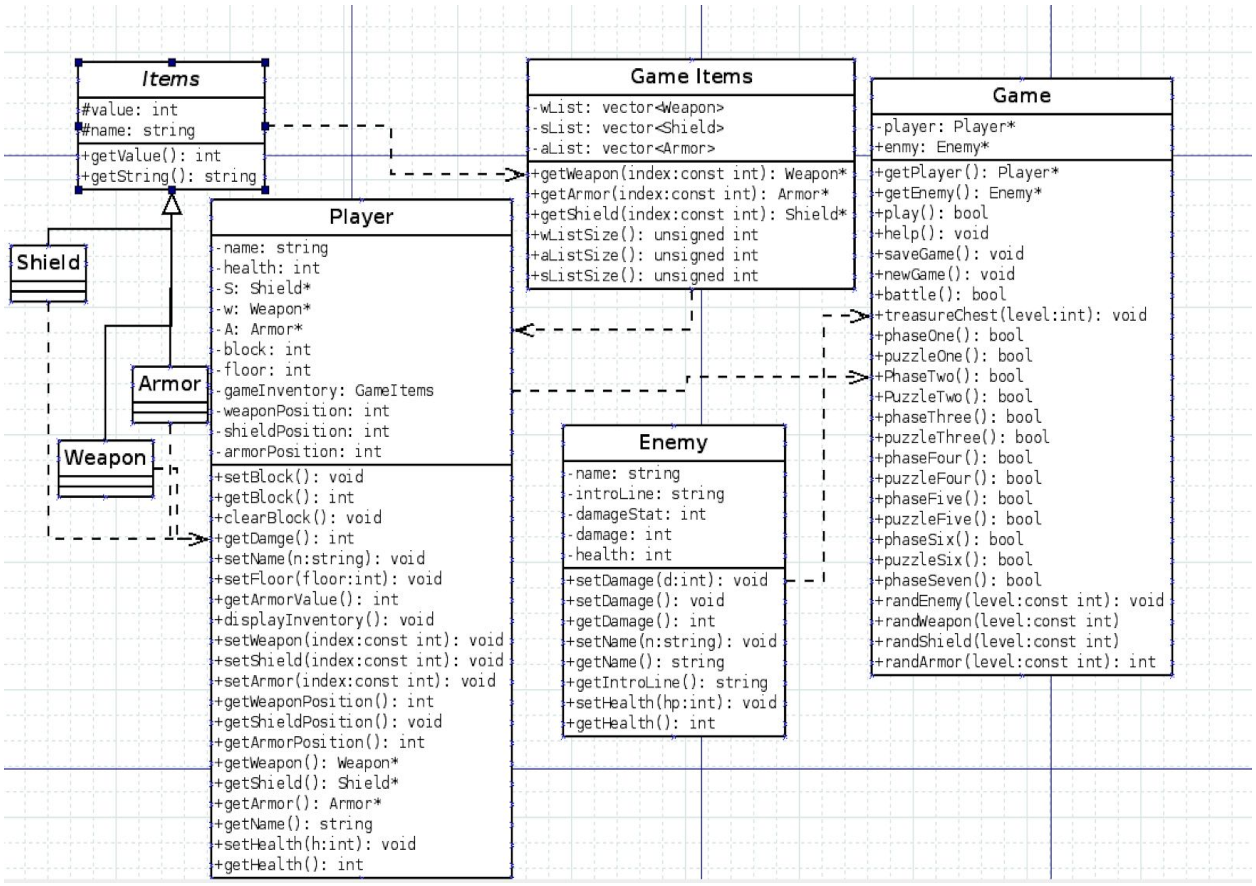
The high level modules in our game is the Game class. Every other lower level module has been geared towards the assistance of the Game class. Items are manipulated through the treasure chest function, the player and enemy classes are used for the battling and the game items uses the weapon, armour and shield classes to store all the weapons.

DRY - Don't Repeat Yourself

We followed the "Don't repeat yourself" principle by abstracting as many functions as we could in the Character and Items classes, reusing as many similar functions as we could. The only unique functions in Weapon, Shield and Armour are the constructors. The Player and Enemy classes contain a lot of similar functions like setter and getter functions for health and name, but were too different to have a base class, so we repeated ourselves a little bit so we didn't violate the liskov substitution principle.

APPENDICES

APPENDIX A: FIGURES AND TABLES


Items
#value: int
#name: string
+getValue(): int
+getString(): string
Shield
Armor
Weapon
Game Items
-wList: vector<Weapon>
-sList: vector<Shield>
-aList: vector<Armor>
+getWeapon(index:const int): Weapon*
+getArmor(index:const int): Armor*
+getShield(index:const int): Shield*
+wListSize(): unsigned int
+aListSize(): unsigned int
+sListSize(): unsigned int
Game
-player: Player*
+enmy: Enemy*
+getPlayer(): Player*
+getEnemy(): Enemy*
+play(): bool
+help(): void
+saveGame(): void
+newGame(): void
+battle(): bool
+treasureChest(level:int): void
+phaseOne(): bool
+puzzleOne(): bool
+PhaseTwo(): bool
+PuzzleTwo(): bool
+phaseThree(): bool
+puzzleThree(): bool
+phaseFour(): bool
+puzzleFour(): bool
+phaseFive(): bool
+puzzleFive(): bool
+phaseSix(): bool
+puzzleSix(): bool
+phaseSeven(): bool
+randEnemy(level:const int): void
+randWeapon(level:const int)
+randShield(level:const int)
+randArmor(level:const int): int
Player
-name: string
-health: int
-S: Shield*
-w: Weapon*
-A: Armor*
-block: int
-floor: int
-gameInventory: GameItems
-weaponPosition: int
-shieldPosition: int
-armorPosition: int
+setBlock(): void
+getBlock(): int
+clearBlock(): void
+getDamge(): int
+setName(n:string): void
+setFloor(floor:int): void
+getArmorValue(): int
+displayInventory(): void
+setWeapon(index:const int): void
+setShield(index:const int): void
+setArmor(index:const int): void
+getWeaponPosition(): int
+getShieldPosition(): void
+getArmorPosition(): int
+getWeapon(): Weapon*
+getShield(): Shield*
+getArmor(): Armor*
+getName(): string
+setHealth(h:int): void
+getHealth(): int
Enemy
-name: string
-introLine: string
-damageStat: int
-damage: int
-health: int
+setDamage(d:int): void
+setDamage(): void
+getDamage(): int
+setName(n:string): void
+getName(): string
+getIntroLine(): string
+setHealth(hp:int): void
+getHealth(): int