Disaster on the Good Ship Lethbridge

Team Ca1rissien

Team C

Max Niu and Steven Deutekom

due February 20, 2019

TABLE OF CONTENTS

Introduction

This is a student game project developed by Max Niu & Steven Deutekom, The following are the purposes of this project:

I. Practice advance object-oriented design and techniques

II. Implement test driven development, error detection and handling

III. Develop team management skill on collaborative software project

IV. Practice UNIX programming tools and scripting language

Disaster on the spaceship Lethbridge is a text based adventure game developed using C++.

The player will start his/her's journey as the captain of the space ship named "Lethbridge" and work their way to solve the puzzles and avoid all the death traps and eventually save the crew and the ship.

There is no scoring. The goal is to reach the end of the game where you save your ship from disaster. Each turn will consist of entering an action and, if necessary, an item to do the action to. The game will process this command and inform the player of the result. This process is repeated until the end is reached or the player is killed.

Player actions

Action 1: Drop — remove an item from players inventory
Action 2: Get — get an item from the environment
Action 3: Use — use an item in the environment or players inventory
Action 4: Talk — interact with an NPC
Action 5: Look — get details about a location or item
Action 7: Inventory — display the players inventory

In game commands

Command 1: Save — save the game
Command 2: Help — display some help documentation
Command 3: Load — load a previously saved game
Command 4: Quit — Leave the game

<Preview of rest of document> The rest of the document proceeds as follows. First, our setup of project management is described. Then, the development process is presented. Lastly, the design goals of the project are outlined. Following this some appendices with class and sequence diagrams are given.

Project Management

This section outlines the team organization. Then gives strategies for risk management of the project.

Team Organization

Max Niu - Design Lead, Story writer, Documentation assistant , Software Developer, Software Tester

Steven Deutekom — Documentation, Quality assurance Lead, Software Developer, Software Tester

Shared Responsibilities - Merge

Risk Management

1. Requirements/Deign/Estimation

Over Estimation: eliminate some functionality, or get basic game working and add what we can in the time we have.

Changes: we do the best we can and save as much as possible but simplify.

Addition: do our best to integrate them and give them tasks that will help finish the game. Could even add some function if having an extra member means we are done quickly.

Technical issues: work together to help understand necessary technical stuff.

Tools all issues: stick to what we know, if things are tough seek help from experienced individuals (teachers/team members).

2.People

Human Casualties: if its close to deadline just finish up what you can, earlier join another group since we are team of two. Press F to pay respect and the name will appear in game credits.

Unproductive team members: team leader will be speak to this individual about the current situation and figure out what caused the loss of productiveness, and put him back on track. note this member must bring snacks to the next team meeting.

3.Tools

Unfamiliar Tools: will not use tool that is not taught in class, ask for help from other member and seek help from instructors.

Development Process

Code Review Process:

this happens every Monday,Wednesday and Friday after cpsc2720 class the two member will meet and discuses the progress they made, if there is any uncertainty on the quality of the code the team lead will decide if we add or reject the code.

Communication Tools:

verbal & telepathic communication as well as text though cell phone.

Change Management:

1.if bug report is filed, the QA lead will find the team member who wrote the bug code and assist them fix the code.

2. if bugs is from two member QA lead will identify the problem and reassign member for bug fix.

3. QA lead will reply to the bug report appropriately.

Software Design

Design:

We tried to use inheritance where possible to encapsulate common behaviors. We also tried to work as many pieces of the game into their own classes with well defined behavior. This also allowed us to separate some components out to their own classes and then build other classes from them. Hopefully, this strategy of inheritance and composition will allow us to write less code and to be able to add functionality where it is needed with ease.

Design Rationale:

The goal of this project is to learn advanced design principals, and to prove we understand these design principals, as well as practice test driven development. We have attempted as stated above to use good judgment in applying inheritance and composition. We are also trying to use the Single responsibility principle where possible. But we recognize that in the projects early stages some of these principals will be violated in order to deliver a product on time. Hopefully, the things we learn in class will help us to improve the project as we go and gain experience from the mistakes we will inevitably make in the early stages.

Update for Implementation Phase(march 16, 2019):

At this time we were able to work with our design relatively well. A version of the strategy pattern was used to implement our event system that allows relatively complex behavior of entities in the game. There are still some bugs with this system and some tweaking is needed to make it more effective. A couple of classes do more than one thing. However we feel that it would not take a lot of work to add more types of behaviors. And the use of inheritance to create groups of behaviors means that even existing behaviors can be composed into more complex ones. Because of time constraints we did not get to fully realize this goal, but we are well on our way

The place were we failed to get good design into our game was with object creation. We have a lot of classes and were forced to save time creating the objects in the game in one function and adding them to a game object. We feel that the builder pattern would be good for our game because some types of objects share a number of common settings. So we could make it possible to create builder objects with these and then use them to make the different objects that share those defaults. It is probably something we will try to work on in the maintenance phase. It would be a real mess to deal with if the game was going to grow.

Also with our System everything is stored with its string name and we feel there are places where if our objects were created with a unique id# that could be used it would be easier to save objects that have similar names. Right now we have a number of objects that should have the same name (specifically doors), but they have to have convoluted names in order to not clash with each other.

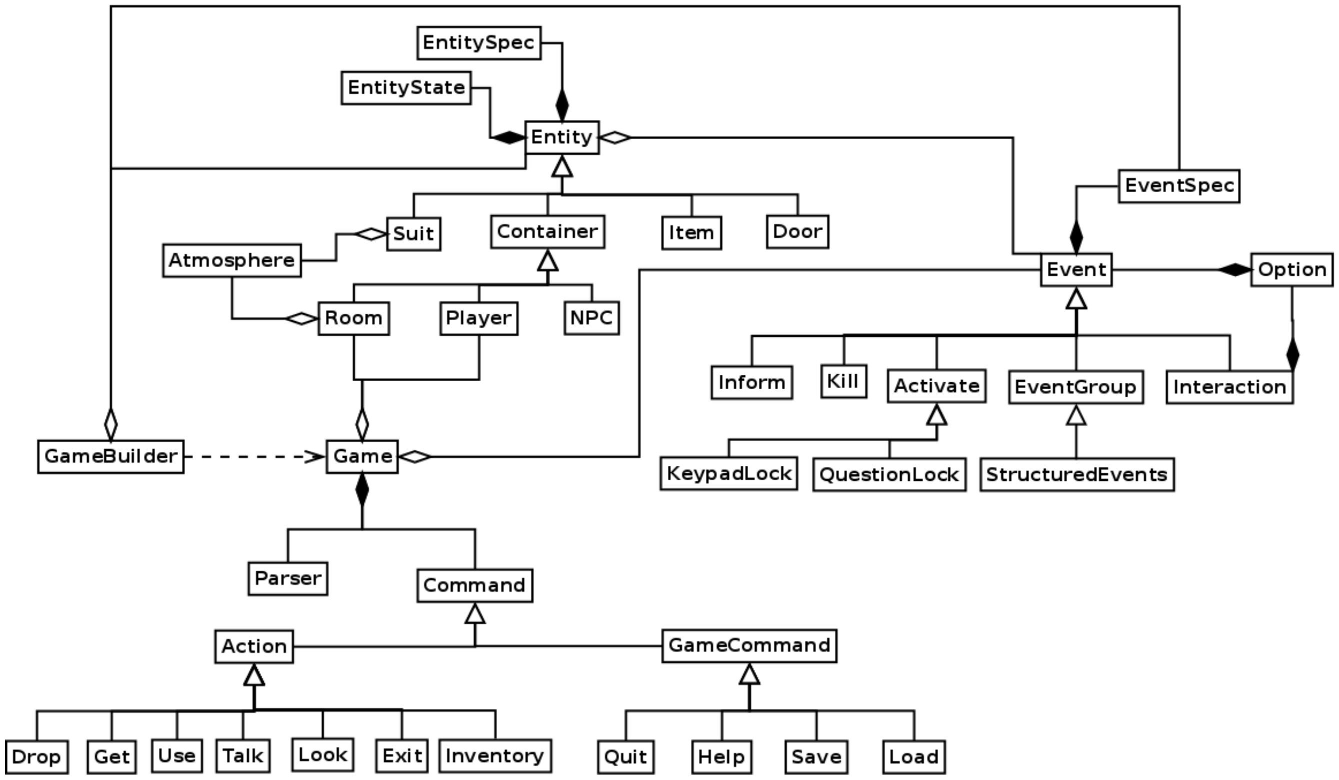
There are more ways we could also continue. The events could be changed to use the observer pattern so that other objects can register to be notified when the events happen. This would make the activate and lock objects work a little better. It would also allow for things like quests to be added. Then all things that want to observe events could have a specific interface and events could activate more than just entities.

If you notice anywhere else where a specific pattern could be helpful please leave us feedback :)

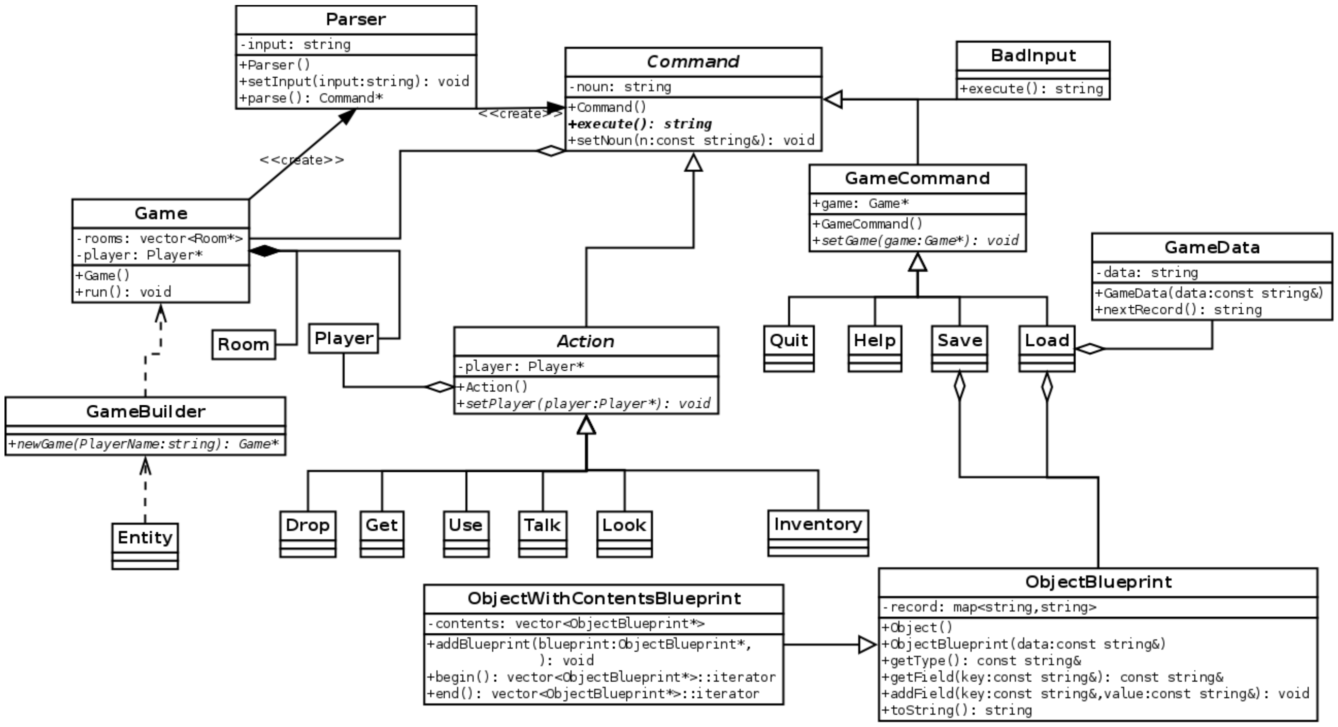
Appendices

Appendix A: Figures and Diagrams

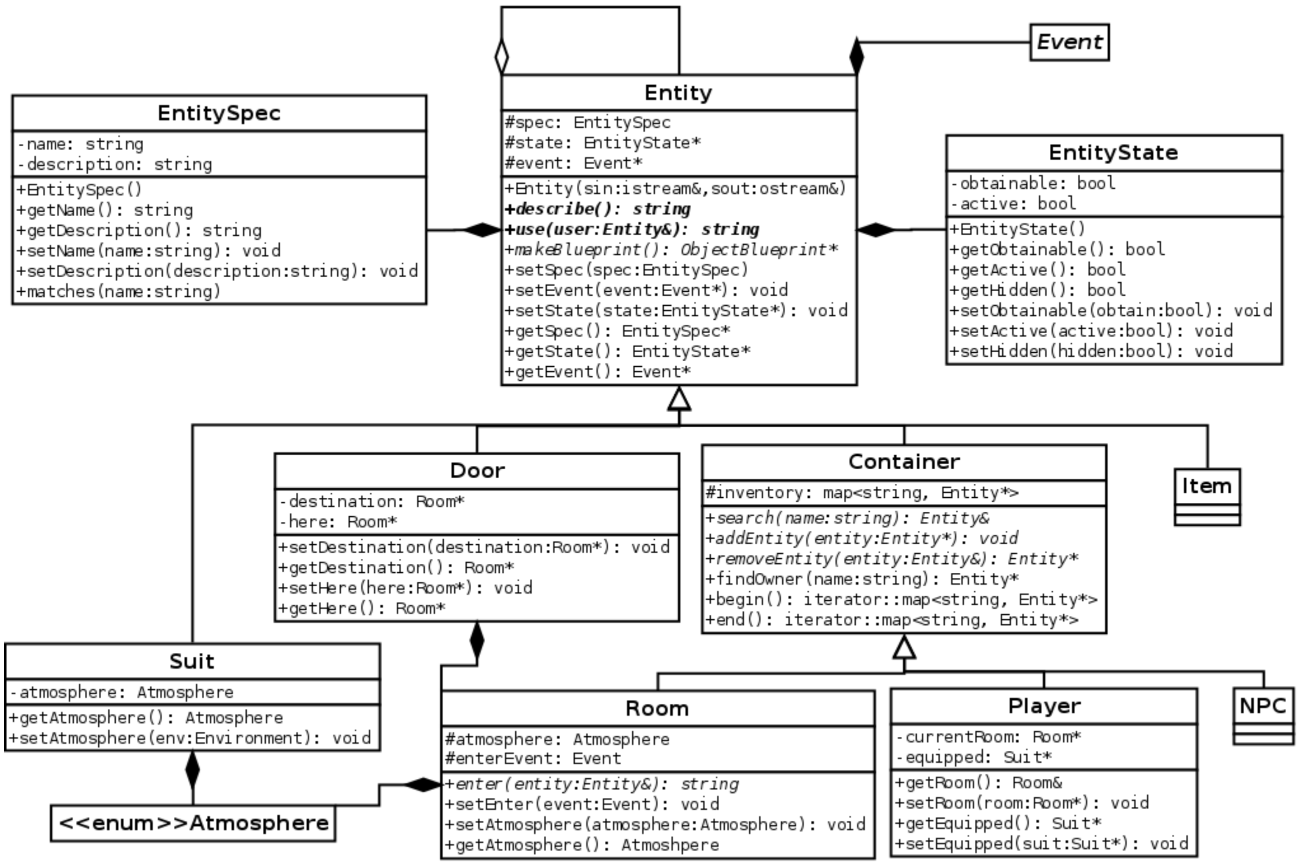
1. Overview (includes all classes currently in the project)



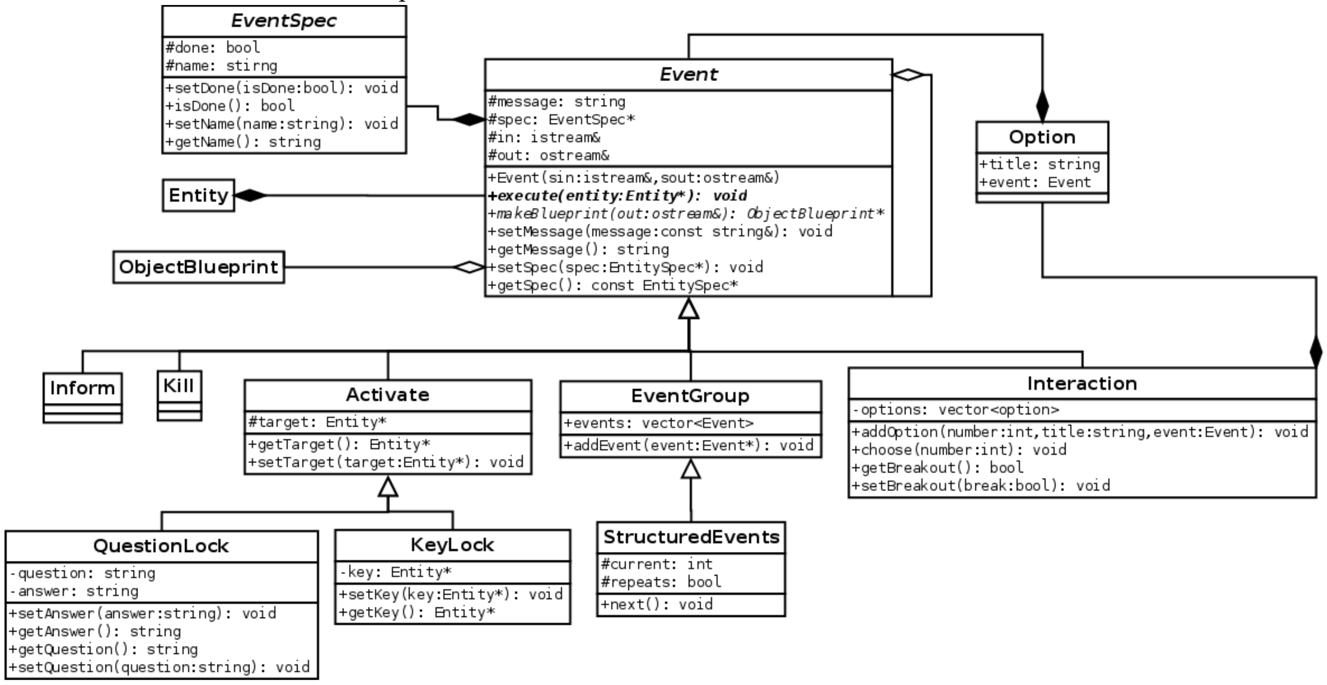
2. Game system components (Room, Player, and Entity on next page)


Parser
-input: string
+Parser()
+setInput(input:string): void
+parse(): Command*
<<create>>
Command
-noun: string
+Command()
+execute(): string
+setNoun(n:const string&): void
BadInput
+execute(): string
<<create>>
Game
-rooms: vector<Room*>
-player: Player*
+Game()
+run(): void
GameCommand
+game: Game*
+GameCommand()
+setGame(game:Game*): void
GameData
-data: string
+GameData(data:const string&)
+nextRecord(): string
Room
Player
Action
-player: Player*
+Action()
+setPlayer(player:Player*): void
Quit
Help
Save
Load
GameBuilder
+newGame(PlayerName:string): Game*
Entity
Drop
Get
Use
Talk
Look
Inventory
ObjectWithContentsBlueprint
-contents: vector<ObjectBlueprint*>
+addBlueprint(blueprint:ObjectBlueprint*,): void
+begin(): vector<ObjectBlueprint*>::iterator
+end(): vector<ObjectBlueprint*>::iterator
ObjectBlueprint
-record: map<string,string>
+Object()
+ObjectBlueprint(data:const string&)
+getType(): const string&
+getField(key:const string&): const string&
+addField(key:const string&,value:const string&): void
+toString(): string

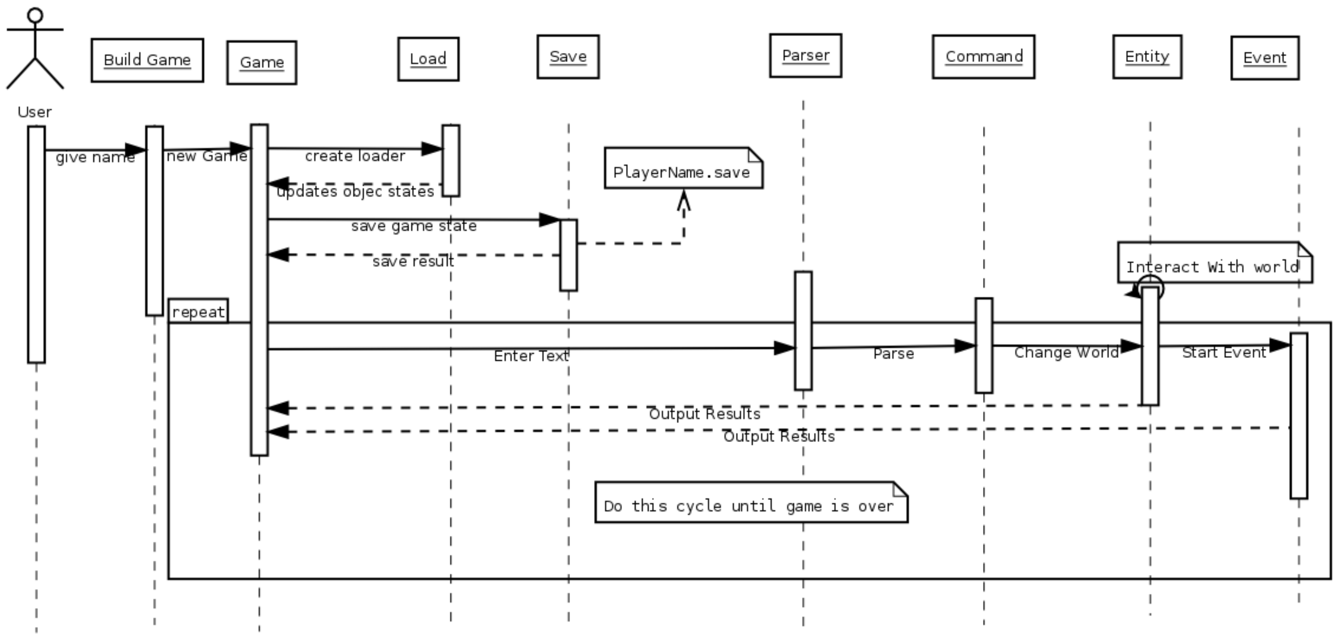
3. Entity components (Menu and Event on the next page)


Event
Entity
#spec: EntitySpec
#state: EntityState*
#event: Event*
+Entity(sin:istream&,sout:ostream&)
+describe(): string
+use(user:Entity&): string
+makeBlueprint(): ObjectBlueprint*
+setSpec(spec:EntitySpec)
+setEvent(event:Event*): void
+setState(state:EntityState*): void
+getSpec(): EntitySpec*
+getState(): EntityState*
+getEvent(): Event*
EntitySpec
-name: string
-description: string
+EntitySpec()
+getName(): string
+getDescription(): string
+setName(name:string): void
+setDescription(description:string): void
+matches(name:string)
EntityState
-obtainable: bool
-active: bool
+EntityState()
+getObtainable(): bool
+getActive(): bool
+getHidden(): bool
+setObtainable(obtain:bool): void
+setActive(active:bool): void
+setHidden(hidden:bool): void
Door
-destination: Room*
-here: Room*
+setDestination(destination:Room*): void
+getDestination(): Room*
+setHere(here:Room*): void
+getHere(): Room*
Container
#inventory: map<string, Entity*>
+search(name:string): Entity&
+addEntity(entity:Entity*): void
+removeEntity(entity:Entity&): Entity*
+findOwner(name:string): Entity*
+begin(): iterator::map<string, Entity*>
+end(): iterator::map<string, Entity*>
Item
Suit
-atmosphere: Atmosphere
+getAtmosphere(): Atmosphere
+setAtmosphere(env:Environment): void
<<enum>>Atmosphere
Room
#atmosphere: Atmosphere
#enterEvent: Event
+enter(entity:Entity&): string
+setEnter(event:Event): void
+setAtmosphere(atmosphere:Atmosphere): void
+getAtmosphere(): Atmoshpere
Player
-currentRoom: Room*
-equipped: Suit*
+getRoom(): Room&
+setRoom(room:Room*): void
+getEquipped(): Suit*
+setEquipped(suit:Suit*): void
NPC

4. Menu and Event components



5. Sequence diagram


User
Build Game
Game
Load
Save
Parser
Command
Entity
Event
give name
new Game
create loader
updates objec states
PlayerName.save
save game state
save result
Interact With world
repeat
Enter Text
Parse
Change World
Start Event
Output Results
Output Results
Do this cycle until game is over