Styx's Treasure Hunt
Analysis and Design Document
Student: Dragoteanu Bogdan
Group: 30_431

Revision History

Date	Version	Description	Author
<17/3/2020>	<1.0>	<First Iteration>	<Dragoteanu Bogdan>

Version: <1.0>

Date: <17/03/2020>

Table of Contents

1. Project Specification

The project is a java application, a game about treasure hunting. When starting a game the user can choose from a few goblin races. These come with racial bonuses and can be customized.

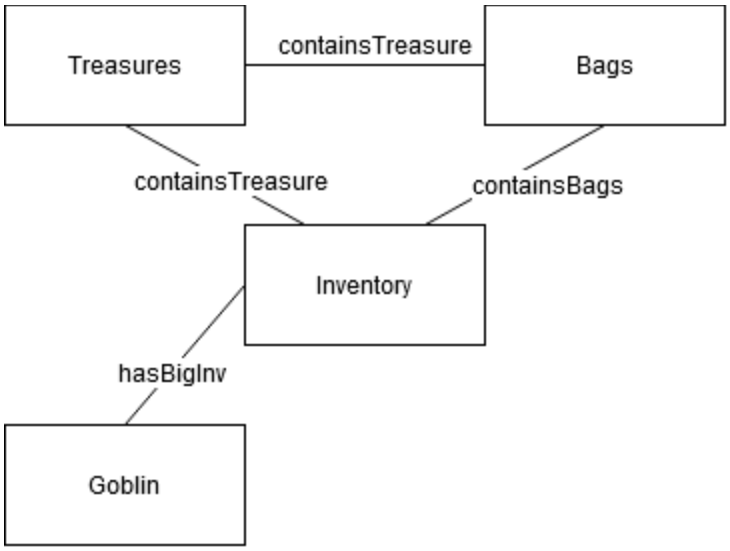
The players go through a series of events, 10 total, which vary in difficulty and size. Difficulty scales depending on the number of guards defending the treasure. Success depends on a series of dice throws simulated by the system, similar to how DnD (Dungeons and Dragons) works.

Once a goblin's inventory fills the user has to pick a bag or he has to progress to the next event without picking up the treasure.

At the end of the challenge goblins are ranked based on the value of the items in their inventory and the winner gets fame throughout the goblin kingdoms.

2. Elaboration – Iteration 1.1

3. Domain Model



4. Architectural Design

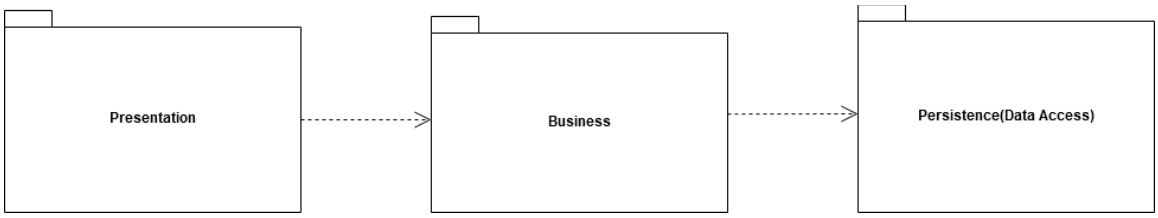
4.1 Conceptual Architecture

- Layered Architecture:
  Components in a layered architecture are organized into horizontal layers, each performing a specific role within the application.
- MVC Pattern → model - contains the core functionality and data
  → view - displays the information to the user
  → controller - handles the input from the user
  → Separates internal representations of information from the ways information is presented to, and accepted from, the user
- Observer Pattern → Used to notify the application that changes have been made
  → Updates the game GUI while the players customize their goblins
- Command Pattern → Used for request execution
- Peer to Peer Pattern → Used for the multiplayer aspect of the system
  → Partitions tasks or workloads between peers.

Version: <1.0>

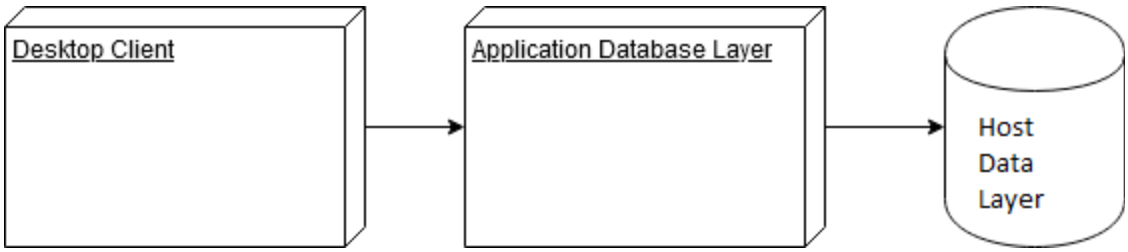
Date: <17/03/2020>

4.2 Package Design(To be updated)

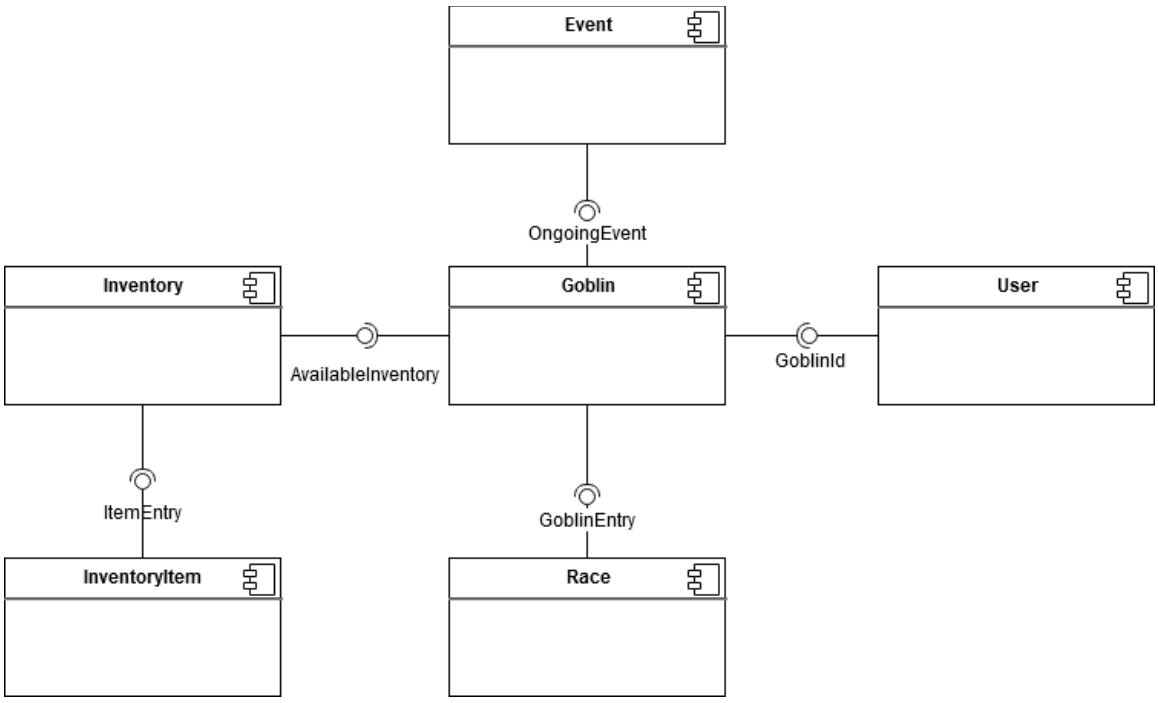


4.3 Component and Deployment Diagrams

Deployment Diagram



Component Diagram

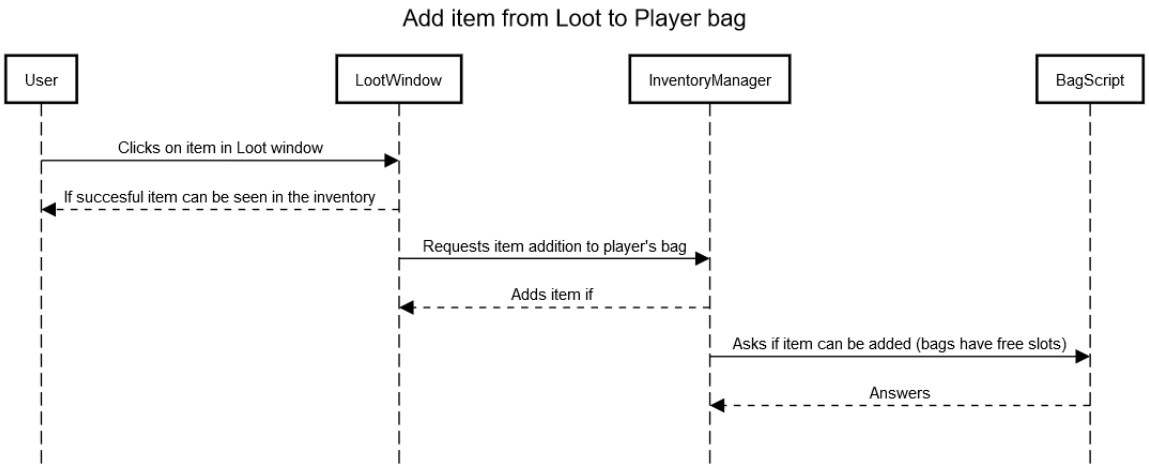


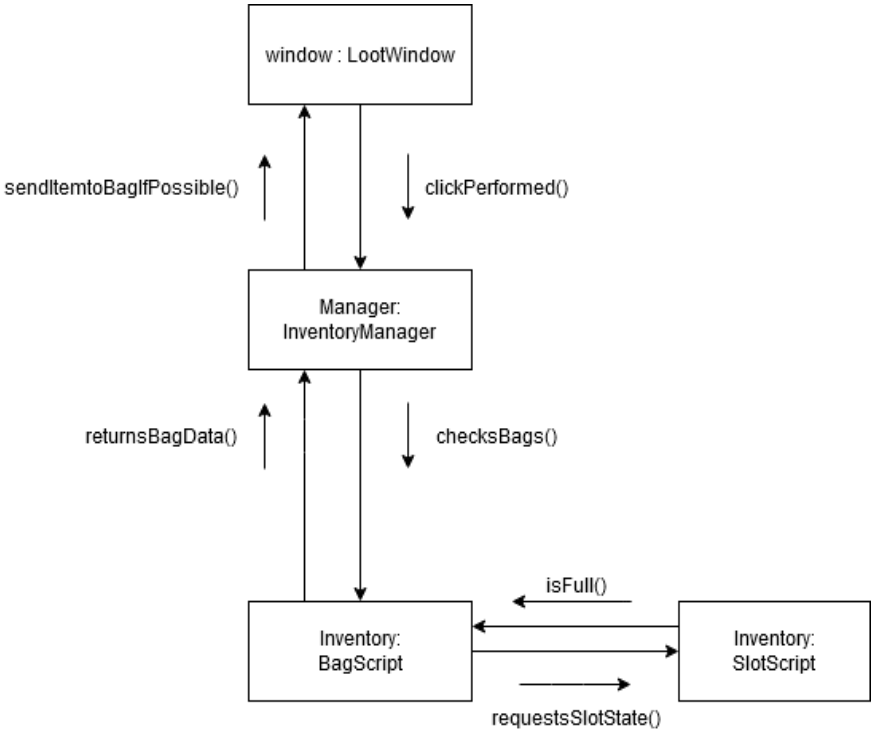
5. Elaboration – Iteration 1.2

6. Design Model

6.1 Dynamic Behavior

Add Item to player bag



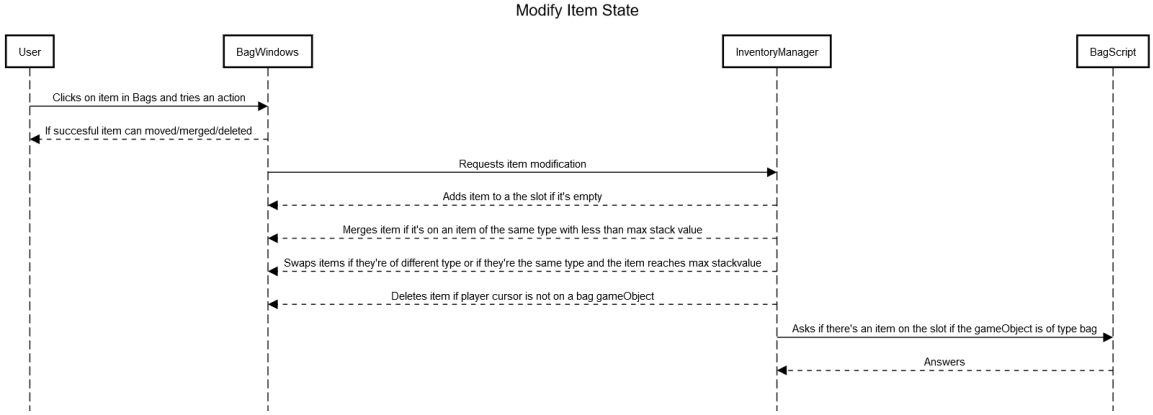


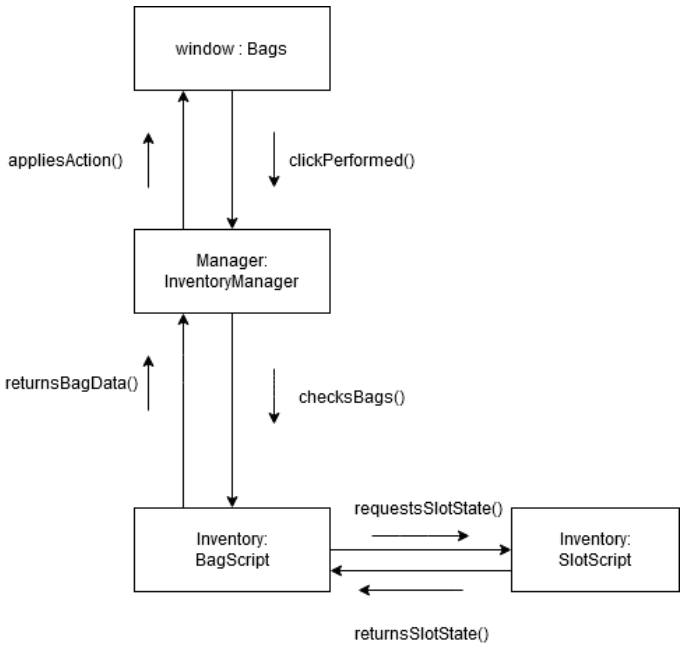
Version: <1.0>

Date: <17/03/2020>

Modify Item State

Items can be moved and merged(if same item type), deleted, swapped.

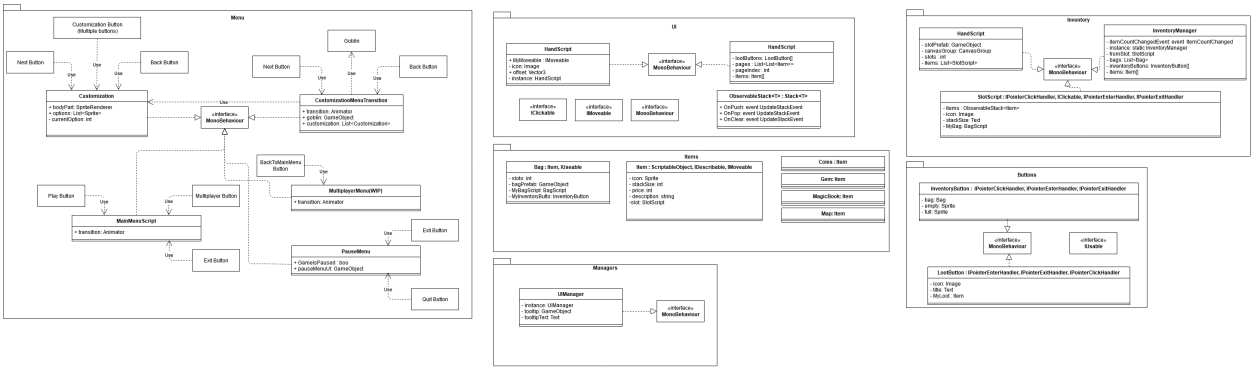


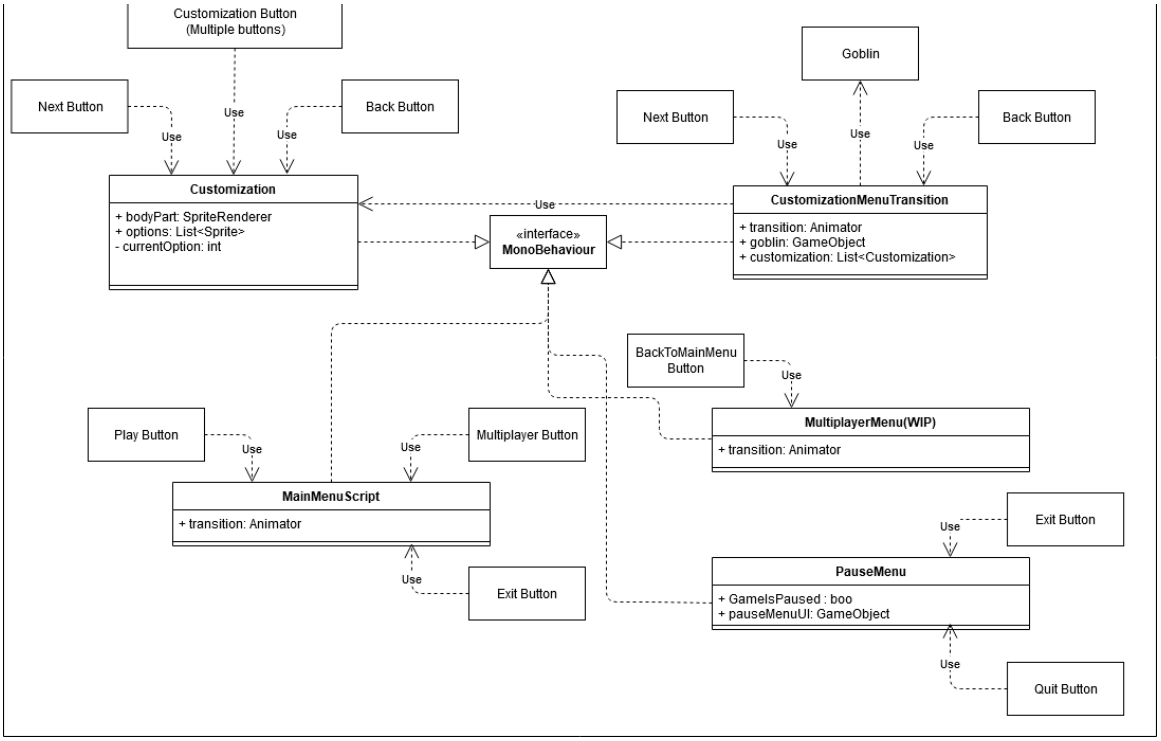


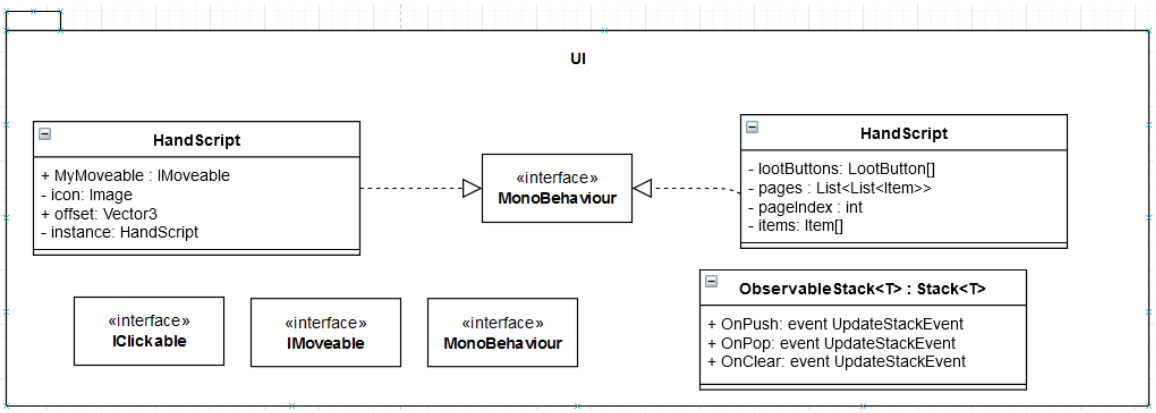
The returnSlotState() is actually more than one function. The system checks if the user moves the item over another slot to apply the item taken on it. The slot is checked for:

- Emptiness
- Ocupied Slot
  → Same item on slot
    → Item is at max stack value
    → Item is not at max stack value
  → Different item on slot
- User pressed on a non gameObject area (this is checkd in the InventoryManager)

6.2 Class Design

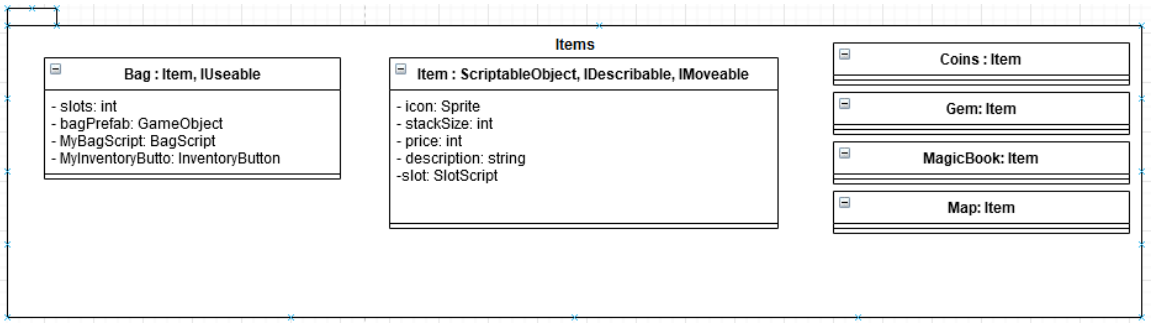


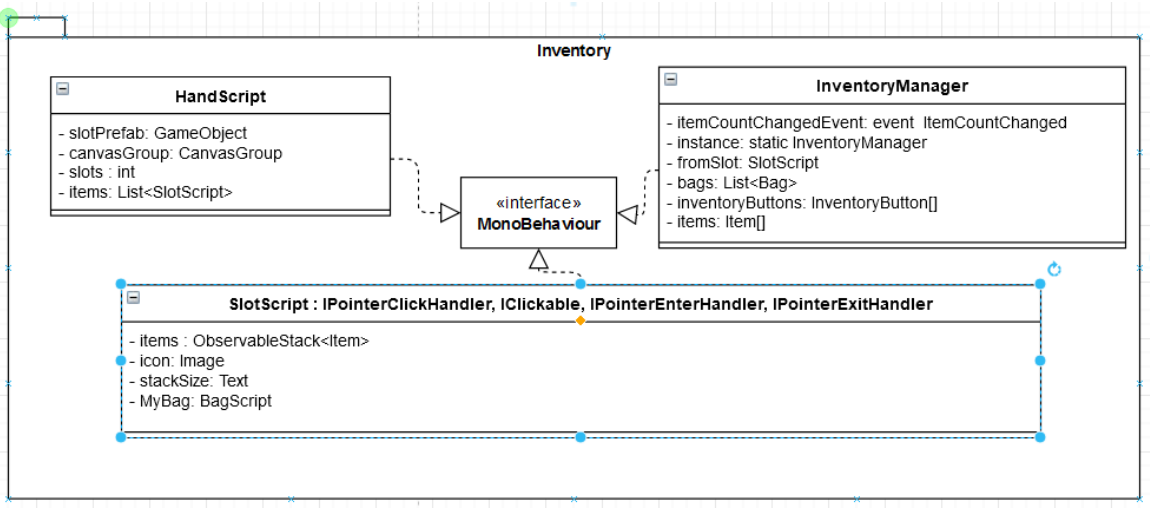


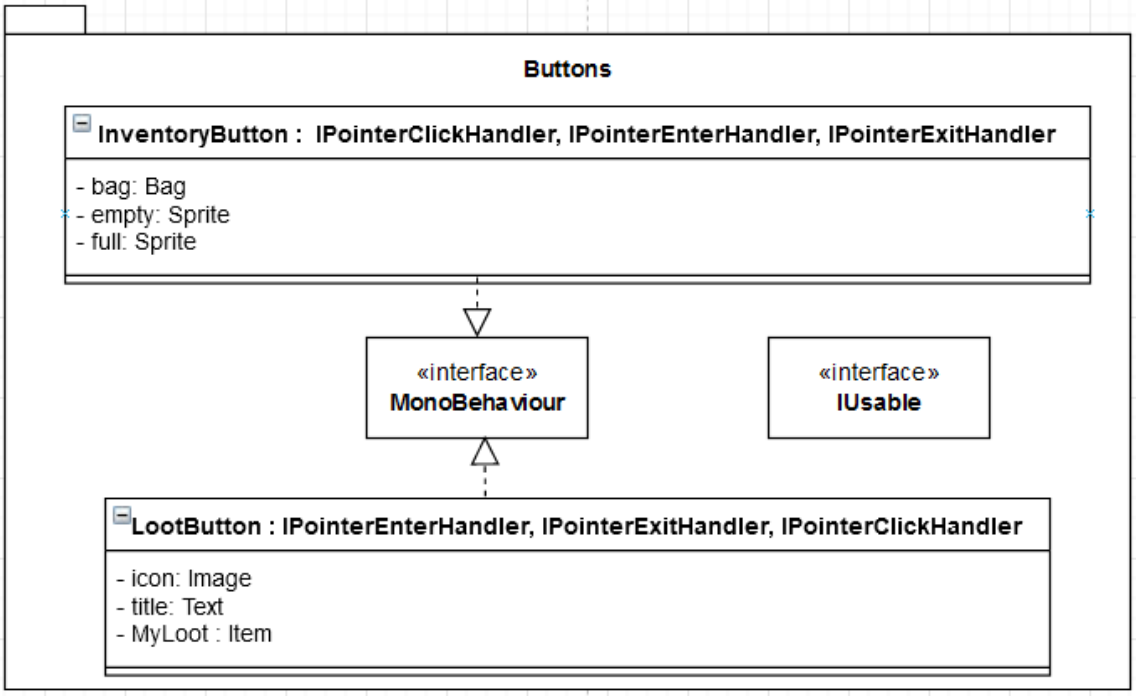


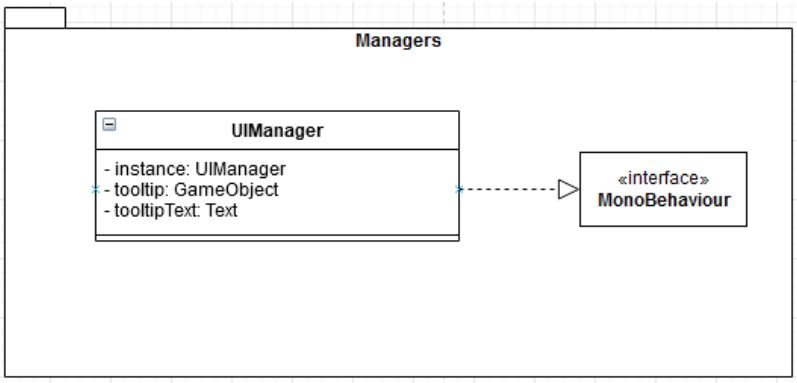
Version: <1.0>

Date: <17/03/2020>





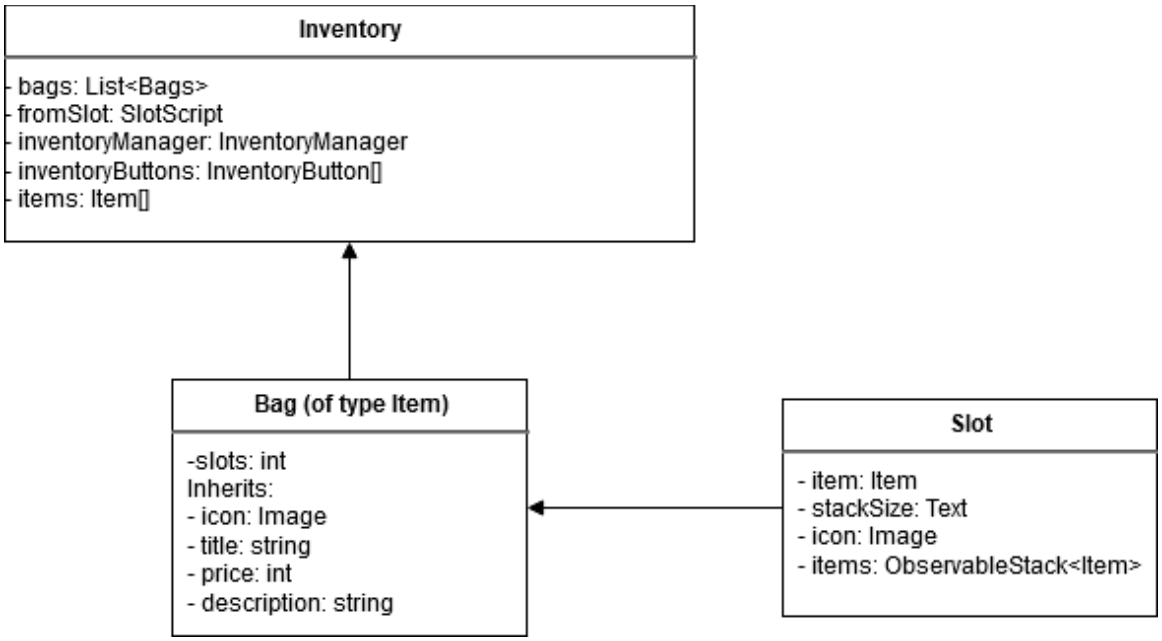




Instead of the implementation arrow naming the implemented class was mentioned in the name in order to make it easier to read on some diagrams.
Some objects weren't connected because I considered that it's intuitive enough so as to know from the 1st and 2nd package images what they represent from the name of the parameters
Observer Pattern → used for item stack check (represented by ObservableStack and called in UIManager)
→ when an item is added to the stack is notified and increments the UI element
→ chosen for easy implementation of a stack system
Singleton Pattern → used to instantiate once certain scripts
→ chosen in order to be sure that certain scripts such as InventoryManager run only in a single instance to not duplicate inventory commands
MVC Pattern → This was chosen as Unity already has some form of MVC in it's structure
→ Easy to troubleshoot

7. Data Model

For the data model I wasn't sure what to make as from what I've seen it's used to show the database so I've shown the model of the player inventory.



8. Unit Testing

[Present the used testing methods and the associated test case scenarios.]

9. Elaboration – Iteration 2

10. Architectural Design Refinement

[Refine the architectural design: conceptual architecture, package design (consider package design principles), component and deployment diagrams. Motivate the changes that have been made.]

11. Design Model Refinement

[Refine the UML class diagram by applying class design principles and GRASP; motivate your choices. Deliver the updated class diagrams.]

12. Construction and Transition

13. System Testing

The testing approach for the host is going to be a top down one. The functionality of the system will be tested from the upper layer to the lowest. For the data receiving, the host will be tested using a bottom up approach.

Using both these approaches → The main approach is something called Sandwich/Hybrid Approach. It's a mix between bottom up and top down approach.

14. Future improvements

> Better models used for the goblins
> New gamemodes
> Better GUI
> Connection Improvements

15. Bibliography

[1] Integration Testiong, http://softwaretestingfundamentals.com/integration-testing/
[2] Design Patterns, Alexander Shivets, https://sourcemaking.com/design_patterns