Tanner Massahos, Joseph Remy, Julian Bell, Tyler Boice, Chase Mosteller

Title of Project: roBOTically efficient

GitHub

D.4 Design – Due: 2018-04-13

CS386 – Software Engineering – Spring 2018

Dr. Marco Gerosa

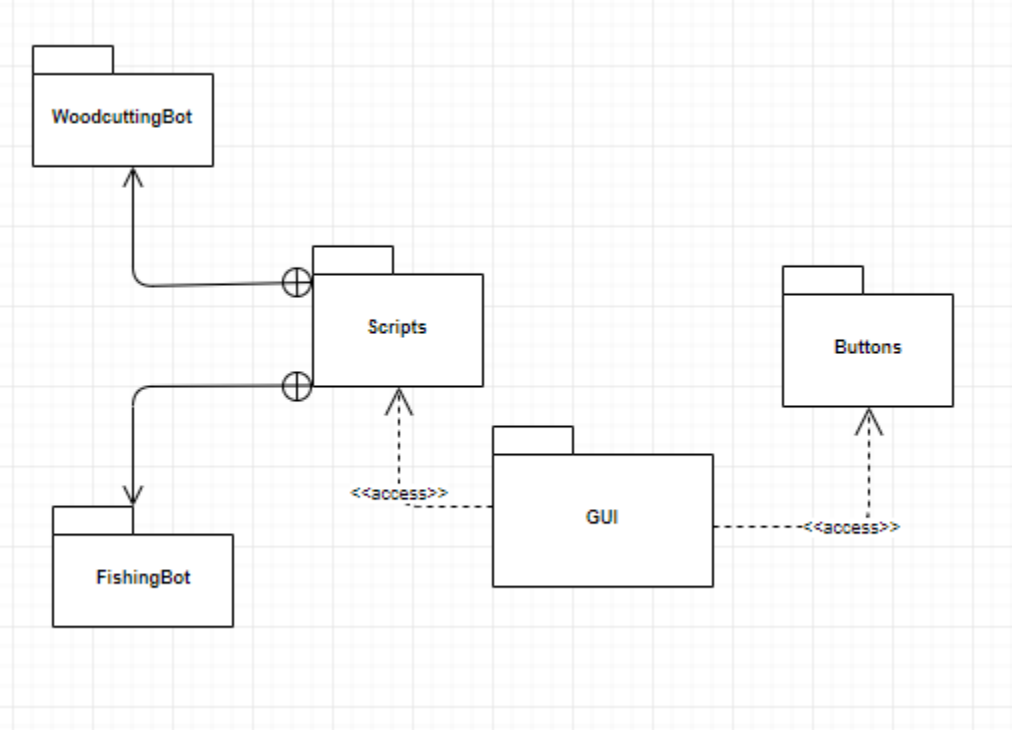
1. Description

Our bot, *roBOTically efficient*, revolves around an Old School Runescape game client a GUI created through Koda to generate a GUI which allows a user to navigate through the script options. The scrips are what do all the work for the user that the user doesn't want to do or doesn't have time to do, these were created by using AutoIt which is a script editor and runner. It also is easily packaged into an executable which is makes it simple to distribute. Currently we have only a woodcutting bot, and have been testing for weeks and have not been banned and gained many levels within the woodcutting skill.

Once the user selects whatever script they would like, it searches for a pixel color and if it finds it, it will click it, chop the tree, drop the logs, and search for the pixel color again.

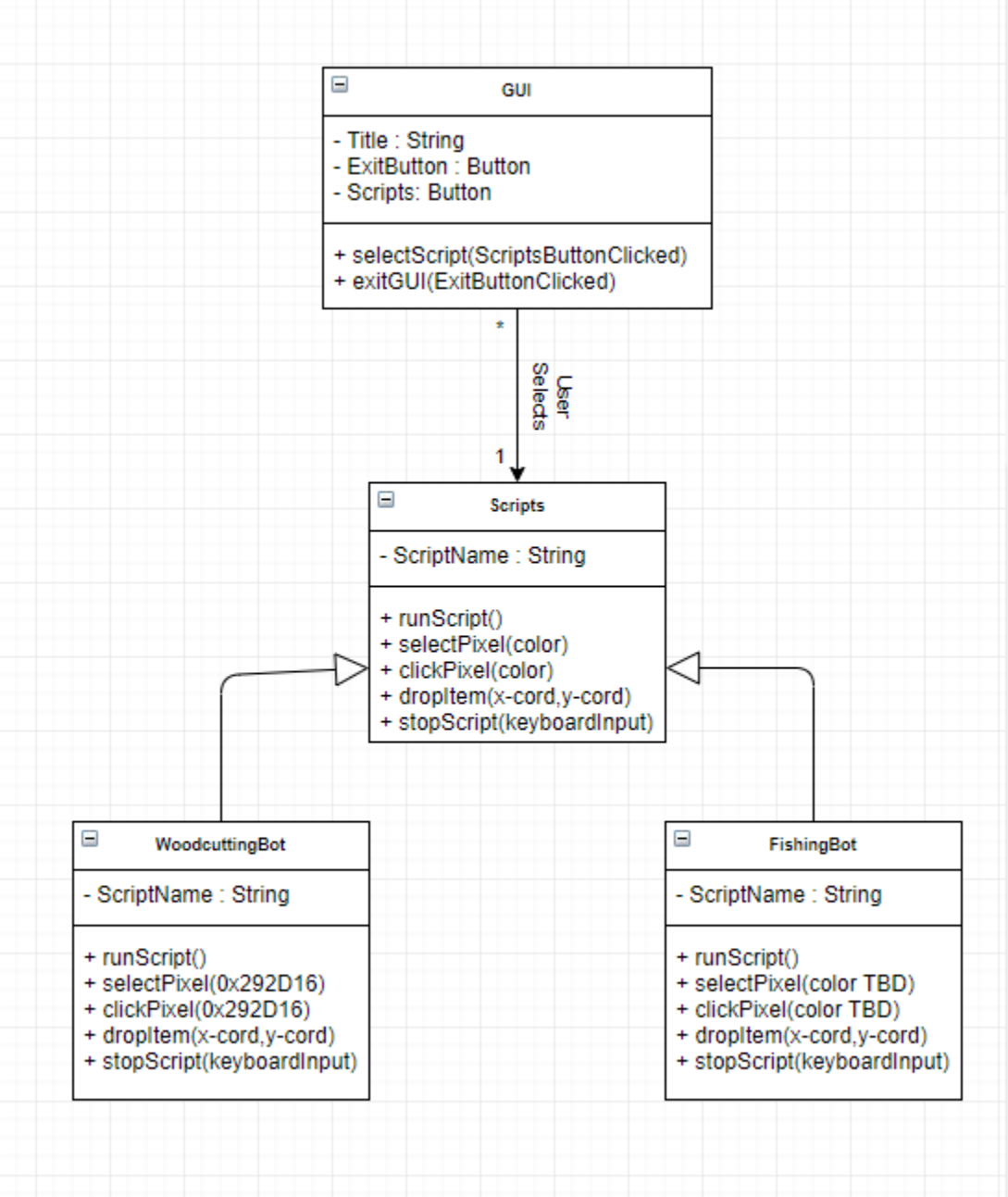
Just a quick disclaimer, since the bots are created through a non OOP scripting language, many of these answers are going to be answered in a way which attempts to fit what the question wants but after looking through these questions it is quite challenging to answer these questions with a non OOP programming language.

2. Architecture



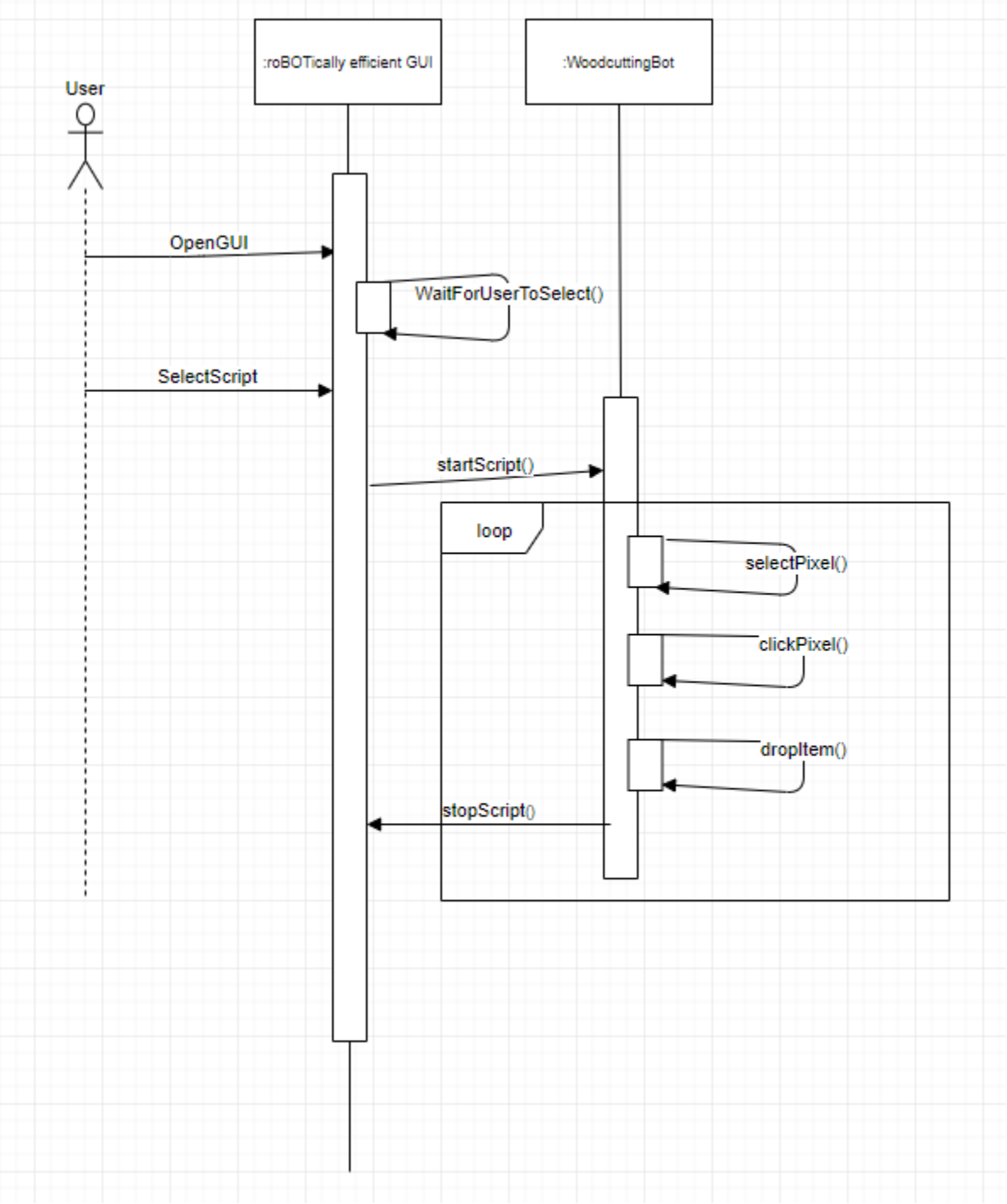
Above is the UML package diagram for our system, it is basically a GUI with buttons and scripts to different types of bots. We figured access made more sense for scripts and buttons as it is more private when compared to import. A user clicks a button which activates a script. It is a bit funky because we used a scripting language.

3. Class Diagram



The current refined class diagram is simplistic as it is simply a user interacting with a GUI in which they can exit the GUI or select a script which has its own unique functionality, which can be seen above.

4. Sequence Diagram



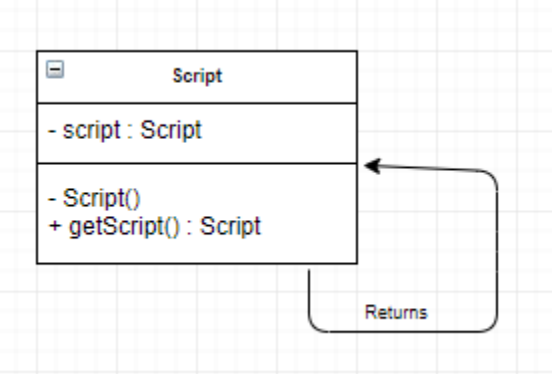
Use Case Description:
The user will open the GUI, upon opening the GUI the user will have the option to select a bot from a list of buttons the user will select whichever script and if they are in the correct location the script will begin doing what is expected infinitely until the user stops the script via a keyboard input. Woodcutting bot in this case.

5. Design Patterns

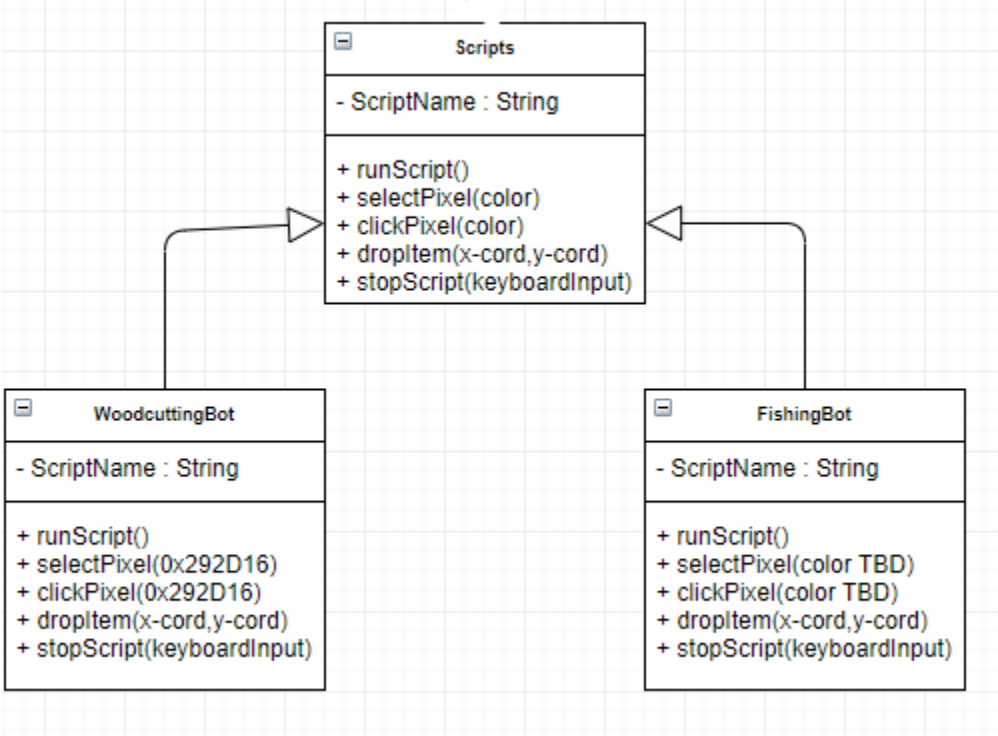
Disclaimer:
Our system does not really fit any of these design patterns so I am going to try my best to make them fit within the design patterns.

All the below “classes” can be found at:
https://github.com/J1411/NAU_CS386/blob/develop/woodcuttingBot.au3

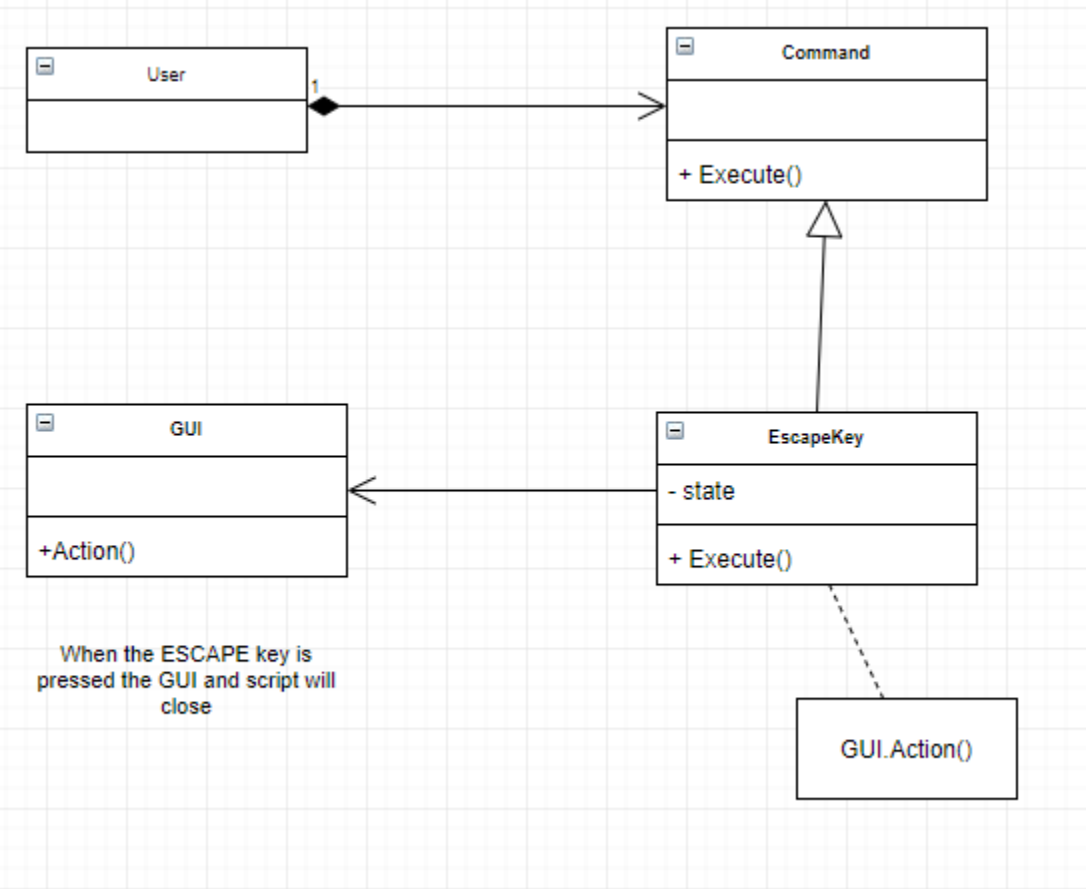
Creational (Singleton Design Pattern):



Behavioral (Strategy Design Pattern):



Behavioral (Command Design Pattern):



6. Design Principles

Disclaimer:
Our system does not really fit the SOLID principles as it is not an OOP programming language. So, it's just one script commented in a way which would make it more OO.

All the below "classes" can be found at:
https://github.com/J1411/NAU_CS386/blob/develop/woodcuttingBot.au3

Single responsibility principle:
All the classes be it GUI or Woodcutting_bot accomplish only a single responsibility. As will any other functionality we add when we further build upon our earlier release.

Open/closed principle:
The scripts we have created thus far are open for extension by the developer but cannot be modified by the user, further behavior can be extended without much change to the source code.

Liskov substitution principle:
I guess our version of this would be the fact that the developer can add more scripts without altering the functionality and correctness of any other script or the entire system.

Interface segregation principle:
This one really doesn't apply to our project as we don't really utilize interfaces at all. But nevertheless, we do split almost everything into smaller and more specific functions rather than one giant function.

Dependency inversion principle:
Our bots don't really depend on much other than if it can't find the pixel which it is searching for. Really doesn't apply to our system since it wasn't created using object-oriented design.

7. Group Participation

Tanner Massahos – Did the class diagram sections and helped anyone who had any questions about what to do on their sections. Also proofread the document and verified that everything was correct. (25%)

Joseph Remy – Created the document, completed the Sequence Diagram and Design Patterns section. Also proofread everything and made sure everything was pretty and presentable. (18.77%)

Julian Bell –. Really rallied everyone and got the group working on this deliverable, also wrote the Description and completed the Architecture section. (18.765%)

Tyler Boice – Assisted in the creation of the document including helping throughout and creating the Design Principles sections. Also was assisting anyone when need be, as we all created this document at once (18.75%)

Chase Mosteller – Assisted where need be and gave solid insight pertaining to a few of the sections people were working on. (18.715%)