Decide-o-matic 10,000
Interim Project Report

Preliminary Requirements, Design, Implementation & Testing

Katabasis

CSC 492 Team 4

Levi Curlee, Minji Kang, Atiya Rulianto, Lillie Sharpe

North Carolina State University
Department of Computer Science

3/07/2025

Executive Summary

Author(s): Levi Curlee
Reviewer(s)/Editor(s): Minji Kang, Atiya Rulianto, Lillie Sharpe

Katabasis is a non-profit organization that aims to help to guide children through various educational challenges. They do this through several different ways, including developing games for students to play.

The problem that we were presented with is this: Students are getting sent out of class for misbehavior, yet some of them do not understand why. This is important to our sponsors because they want students to be able to learn and to stay in class, so knowing why they get sent out of class is a great way to help them. This is especially helpful for students who have emotional and behavior disorders (EBD).

Our solution to this problem is to make a game where students are able to learn how their actions affect those around them and how they result in getting sent out of class. This is done through the users playing a game where a robot, "Steve", is helping to guide the student through what happened in class. Steve will be guided through what the student did to get in trouble, what happened after, and the outcomes that were affected. These will all be seen as different doors that the user can guide Steve to. This will then be visualized to the student using a Causal Decision Diagram (CDD). The goal of this CDD is to tell the student how their actions impacted the result of them getting sent out of class, their friendships, and more.

The requirements that we have are mainly scene-based. They include functionality with doors, jumping from different platforms, allowing students to input text, allowing the CDD to be edited, and allowing students to loop back to the beginning if they got in trouble for multiple misbehaviors. We have designed a paper prototype, wireframes, UML class diagrams, and a high-level diagram. Along with this, at the time of this submission we have successfully implemented scenes 1-5, along with scene 7 (the CDD scene). Some of these scenes are not fully done being implemented, however they are currently playable.

We have also been able to test a lot of the game. Since this is a game, it is difficult to unit test, however we have made a lot of system tests. The game startup, movement, door interactions, and candy collection tests are all passing. The obstacles, text inputs, and CDD display scenes have some passing tests and some failing tests. The CDD editing and multiple playthrough/restart tests are all failing. Additionally, we have completed the first iteration of usability testing and have received great feedback which we are going to be implementing from now on.

Our next steps include updating the code to reflect the feedback from the usability tests, creating more scenes, and cleaning up the UI to make the game more cohesive.

Project Description

Author(s): Lillie Sharpe
Reviewer(s)/Editor(s): Atiya Rulianto, Minji Kang, Levi Curlee

Sponsor Background

Katabasis is a nonprofit organization located in Grifton, NC that strives to provide children with the guidance and support that they need to succeed in their educational journey, especially children who come from low-income backgrounds. Currently, they are focused on developing smart programs to assist the learning of children at the elementary and middle school levels. They aim to boost the relationship between these children and their teaching staff by creating understanding between them. Funding will be sought from donations and grants for their projects, and to promote their projects, they plan to coordinate with educational institutions and digital creators.

Name:

Katabasis

Title:

Organization

Address:

900 Riverside Rd
Grifton, NC 28530

Website Address:

https://www.katabasis.org/

Sponsor Contacts:

Bill Causey
President
bill.causey@katabasis.org

Dr. Joseph B. Wiggins
Vice President
joseph.wiggins@katabasis.org

Jorge Parra
Technical Lead
jorge.parra@katabasis.org

Alex Cail
Software Developer
alex.cail@katabasis.org

Ben Taylor
Software Developer

ben.taylor@katabasis.org

Problem Description

The problem that our sponsor has presented our team with is that students, especially those with emotional and behavioral disorders (EBDs), might not understand the connections between the actions that they make and their consequences. Because of this, they might make poor decisions without realizing that they are doing something wrong or understanding the impact that those choices can have on their lives. These students might be frequently sent out of the classroom if they regularly make poor decisions and get in trouble in school, but they may not learn from or respond well to the punishments that they receive. If these students could be educated about decision making in a way that engages them and is easy for them to understand, then not only would they have less difficulty in their social and academic lives, but the people around them would no longer be affected by any negative consequences that arise from their poor decisions.

With our project, we hope to address this need by creating a low-specification browser game that will help these students learn the consequences that their actions have on the people around them and how they can make more good decisions in the future. By making our game a browser game that has low-specification requirements, we are able to follow the goals of our sponsor to make education and learning more accessible for all students. We will make our game easy to understand and play for elementary and middle school students, and we will use a causal decision diagram (CDD) to help them understand how actions and decisions impact their environment.

Proposed Solution & Project Goals/Benefits

The proposed solution that our sponsor and team has devised to address the above problem of students, especially those with emotional and behavioral disorders, not understanding the connections between actions and their consequences is to create a low-specification browser game. A game will engage students so that they will more effectively learn good decision making skills while also wanting to learn about them. Since our audience will be elementary and middle school students, we will make our game easy to understand and play, and upon the requirement of our sponsor, this game will also include a causal decision diagram to give the students a visual aid for the cause and effect relationships of the actions they make and their consequences. By creating our game to be browser compatible and require low computer specifications, it will be able to be effectively run in schools that may not have the hardware to run demanding software. This will allow more students from different backgrounds to benefit from the decision making education made accessible through the game, aligning with the mission goals of our sponsor.

For the clientele of our sponsor, this game will be a great resource for students and teachers, especially for those from low-income backgrounds. Students, especially those with emotional and behavioral disorders, may not understand how their actions affect the people around them and why negative consequences might arise from doing negative actions. With our game, teachers will have a tool that they can use to teach students who are frequently sent out of the classroom how their actions relate to their consequences. This will not only improve their social and academic skills while also providing them much needed relief through education, but will also provide relief to their teachers and the people around them. This aim firmly aligns with the goals of our sponsor to make education more accessible to all people.

Resources Needed

Author(s): Minji Kang
Reviewer(s)/Editor(s): Levi Curlee, Atiya Rulianto, Lillie Sharpe

Resource Name	Purpose	Status	Version	Licensing Information
Phaser	JavaScript library to create games	Obtained access	3.87.0	MIT License
OpenDI Causal Decision Diagram (CDD) Authoring Tool	Required tool to generate the Causal Decision Diagram created by player choices	Obtained access	Prototype	MIT License
npm	Manages packages and dependencies for JavaScript	Obtained access	11.0.0	The Artistic License 2.0
webpack	Bundles JavaScript files	Obtained access	5.97.1	MIT License
jointJS	Creates the Decision Diagram UI	Obtained access	4.0.2	Open Source Mozilla Public License Version 2.0.

Risks & Risk Mitigation

Author(s): Lillie Sharpe, Atiya Rulianto
Reviewer(s)/Editor(s): Minji Kang, Levi Curlee

- **Outlier Cases**: With the way our game design is structured, we will have a common flow of a student's given situation of why they may have been sent out of class. However, there are rare instances where we cannot predict a student's issue or situation through the dialogue options of our game. For example, due to a lack of available choices within our game, if a student was affected by an outside factor or extremity when they supposedly performed the action that got them sent out of their classroom, there is no dialogue choice for the student that represents the existence of the external.

    - *Mitigation*: To counteract this risk, we are incorporating text inputs for students to thoroughly describe a situation that is not covered by the given paths our game provides. Also, we are also adding other text box interfaces in later scenes that will allow students to indicate whether or not there were external influences on the situation that they got in trouble for, or potentially for their further thoughts about the situation. This could give teachers more insight into what actually happened in the situation.
- **Player Honesty**: Our target audience is middle school students. From our personal experiences and what we know of middle schoolers, they have a tendency to try to outsmart others and lie about some of their misbehaving actions. They might try to answer “I don’t know” when prompted to give a reason to why they made the player character choose a certain action.
    - *Mitigation:* Our team is aiming to solve this problem by using the player character, Steve, as a friendly guide that they can be honest with and have more conversational dialogue with text input prompts. We may try to later implement a flagging system that can prevent the user from typing random characters to bypass any text boxes.
- **Player Understanding**: Similarly to the Player Honesty risk above, these students may also genuinely not understand that what they did was wrong if they did commit the action that got them sent out of the classroom. Instead of responding to questions with “I don’t know” because they are attempting to outsmart the game that we are creating, they might respond with that statement because they genuinely do not understand how what they did was wrong.
    - *Mitigation:* To avoid this, our team plans to do our utmost to use wording throughout our game that students can easily understand. By including a button to indicate when a student is confused during a text prompt, we can reiterate the question posed and give the student further understanding of what is being asked of them. If a student is constantly answering “I don’t know” for multiple text inputs, we can use this information to alert the teacher and have them elaborate what must be done for the student. This will hopefully educate them more about their decisions and allow them to complete the game more effectively.
- **Computer Specs**: When creating an online game, something we must consider is the technical specifications of one’s device to access the project. Low-spec computers have a harder time running modern software programs thus can cause low-quality graphics, longer load times, lower frame rates, and glitches.
    - *Mitigation:* To mitigate this risk, our team chose to code the video game using Phaser that has an automatic configuration to WebGL on all devices. WebGL is a specific API for Javascript that allows for 2D/3D graphics to run on any web browser without the use of plugins. By using WebGL, we guarantee that our game can run on any web browser regardless of the specs on a student’s computer.
- **Lack of Interest**: A risk that comes from any game designed for young students is how much they actually want to participate and play the game. With our target users being middle schoolers, it can be easy to bore the student to not want to continue with the program if it’s too repetitive, dull in graphics, or just overall uninteresting. Our team runs

a higher risk with this issue since we don't have a reward system. The students need to be engaged enough to follow the gameplay but not too invested that they would purposefully be sent out of class just to play.
    - *Mitigation:* The first step to mitigate this risk is creating colorful assets and graphics that look appealing to the eye. The second step is to run usability tests with multiple users to get feedback and see what can be improved. Taking both steps into account, we can minimize the likeliness that a player will lose motivation when playing our game.
- **Accessibility**: To our team, providing features that would assist the fun and understanding of students who might have accessibility needs is also important to the goal of our game. Because of this, we have been considering the possibility of colorblind students playing our game. One of our team members grew up with a good friend who is colorblind, so they know what a colorblind person would be able to see better in comparison to what they might not be able to see as well.
    - *Mitigation:* With their guidance, we have made our interactable assets stand out against our backgrounds to ensure that students who are colorblind can also see obstacles and goals better. In most variations of colorblindness, colorblind people are unable to see green, red, or both. Instead, green and red look similar to yellow or brown to them. We have made our obstacles indigo or purple to make them stand out more against the green and pink backgrounds that we have created, and we have made our goals blue with yellow details to make them stand out more. The doors are orangey yellow with highly saturated blue edges and blue doorknobs, and the candies in the candy level have blue wrappers tied with yellow strings.

Development Methodology

Author(s): Atiya Rulianto
Reviewer(s)/Editor(s): Levi Curlee, Minji Kang, Lillie Sharpe

For the development process of our project, we have decided to work based on scene based sprints. Because our project is a game, it is split up into different levels, or scenes as described by our game development library, Phaser. Our team has weekly discussions at the start of the week during class on what scenes we should be focusing on and the aspects/concepts that should be implemented or improved. During the week, multiple people can be working on the same scene while other group members are working on another. By the end of the week, our team is expected to have some functionality in the program and have logs of what code snippet they worked on. These contributions should be shown by the start of the next in-person meeting before discussing next step plans. We utilize Github for creating branches for every new feature we are working on during a sprint so there are less merge conflicts on the main or development branch. However, our focus may be shifted into working on other parts of our project like reports and presentations. For the first few weeks of the senior design course, our main focus was on design and researching. These early iterations did not include any scene development or testing. However, after implementing more scenes, our team has been running both usability tests and system tests on the project based on the functionality we have completed for the weekly sprint. Every Wednesday, our team meets with our sponsors and discuss a specified topic that is indicated on the introduction packet we received at the beginning of the semester. Our weekly sprints are also based on what is expected for us to show to our sponsors.

System Requirements

Author(s): Levi Curlee, Atiya Rulianto
Reviewer(s)/Editor(s): Lillie Sharpe, Minji Kang

Overall View

The users of this program are middle school students who got sent out of class for their misbehavior. The user must help the robot (Steve) to describe what happened that day. This is done mainly through doors, but also with typing. The user will guide Steve to different doors that best describe what happened in scenes 1 and 2, then will type out in scenes 3, 5, and 6. The user will pick up various candies in scene 4 in order to select which outcomes were affected. Lastly, the game will show the user the CDD of the different choices that they made. This will meet the need of helping users connect their actions with the results of their actions.

Functional Requirements

GENRE: 2D, choice-based adventure game

PLAYER: Middle School Student

GOAL: Learn the consequences of their real-life actions (student) and learn about decision paths/decision understanding (student and teacher) **[scene based goals]**

Scene 1 – Free Move:
- [GR 1]: Player must enter one of the doors presented
- [GR 2]: Player should be able to move in any direction
- [GR 3]: Player must navigate around obstacles
- [GR 4]: If player does not select one of the three doors, a fourth door with a text input option appears

Scene 2 - Platforms:
- [GR 1] applies
- [GR 5]: Player will have gravity applied and can jump between platforms

Scene 3, 5, 6 – Intermediates & Externals:
- [GR 6]: Player must log a text input for the Causal Decision Diagram

Scene 4 - Outcomes:
- [GR 2] applies
- [GR 7]: Player must choose at least one outcome

Scene 7 - CDD:

- [GR 8]: Causal Decision Diagram should display player choices
- [GR 9]: Diagram should have descriptions for easy understanding.
- [GR 10]: Player should be able to edit their Causal Decision Diagram.

Post Scene - Restart:
- [GR 11]: Player must have the ability to loop to the start
- [GR 12]: Player must run through a good example

Non-Functional Requirements

[NF 1] A single full playthrough shall last at least 5 minutes.

[NF 2] A single Causal Decision Diagram path should contain 6 scenes (not including the Causal Decision Diagram Scene).

Constraints

[C1] The system must use the OpenDI CDD Authoring Tool.

[C2] The system must be compatible with WebGL.

[C3] The system must be a web browser game.

[C4] The system must be "gamified" to go beyond simple text dialogue.

Design

Author(s): Levi Curlee, Minji Kang, Atiya Rulianto, Lillie Sharpe
Reviewer(s)/Editor(s): All

High-Level Design

For the Decide-o-matic 10,000, our target users are middle school students who may have emotional or behavioral disorders. These users will interact with the frontend of our project. The game will take in user inputs and react accordingly with its gameplay and usability. This frontend is built with scene files using Javascript and PNG assets for accessibility and artistic designs. The majority of our game is implemented using basic mechanics from the Phaser library. This includes player movements [GR 2], gravity aspects [GR 5], and text inputs [GR 6]. However, for the last scene (Scene 7), our team utilizes the CDD Authoring Tool created and developed by OpenDI. Any decisions a player makes during gameplay will be saved and sent to the CDD Authoring Tool. The tool returns a corresponding graph component such as a lever, intermediate, or outcome. This specific component fulfills requirements [GR8] and [GR9]. These graph components should display once the player has reached the final scene of the game. The game is currently being built and tested using Webpack and npm.

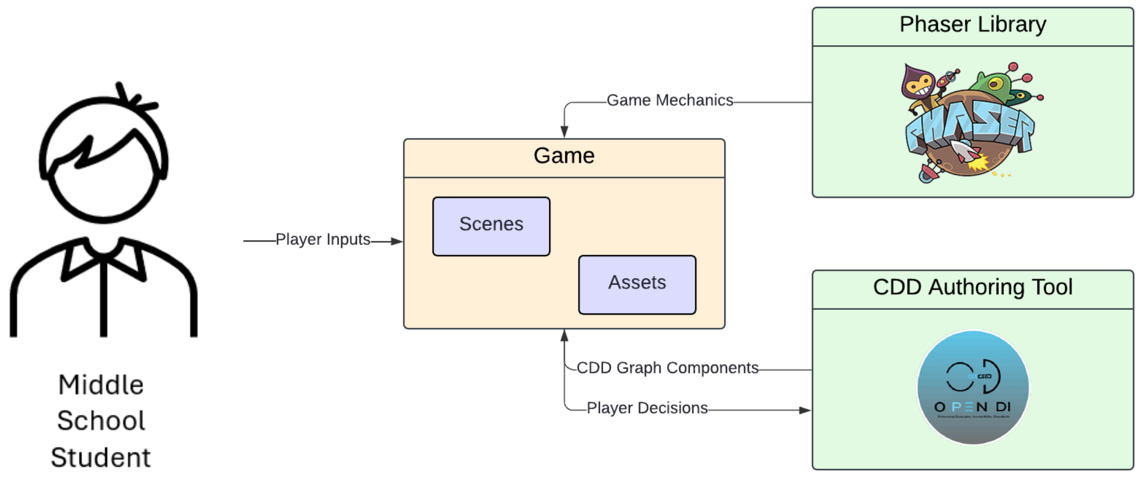


Figure 1. High Level Design Diagram

Low-Level Design

For our low level design, our team focused on creating a UML class diagram to describe the project. The main system that users will interact with is the Main class. This class configures the game with the config variable and stores it in the game variable. In order to start the project, the game variable must add each scene provided from the Scenes folder and choose the given start scene. For our project, we start with the Title Scene class. The arrow connecting the Scene folder to the Main class represents the scenes being added and displayed to said class.

Each Scene class has three common functions provided by the Phaser library: preload(), create(), and update(). Preload() is used to load in variables, images, and assets into a given scene. Create() runs at the beginning of the game to make objects, text, animation, sprites, etc. that have been preloaded into the system. Finally, update() runs continuously and will allow different variables to change throughout the game. Scene 1 and Scene 2 both have functions for the door functionality of our game [GR 1] and Scene 4 has a function for candy pickup [GR 7]. Most of the variables described in each class are the player sprite, the movement cursors/inputs ([GR 2], [GR 3], and [GR 5]), and the player's text input they may enter ([GR 4] and [GR 6]).

The CDD Scene interacts with the OpenDI CDD Authoring Tool by using the functions defined in the index.js in the Authoring Tool. It listens for player decisions in the background and creates the CDD levers, intermediates, externals, outcomes, and dependencies using the functions in index.js. The functions that create the CDD use the graph components called Decision Elements and Causal Dependencies which are JavaScript files that allow for the addition of the elements and dependencies onto the JointJS graph.

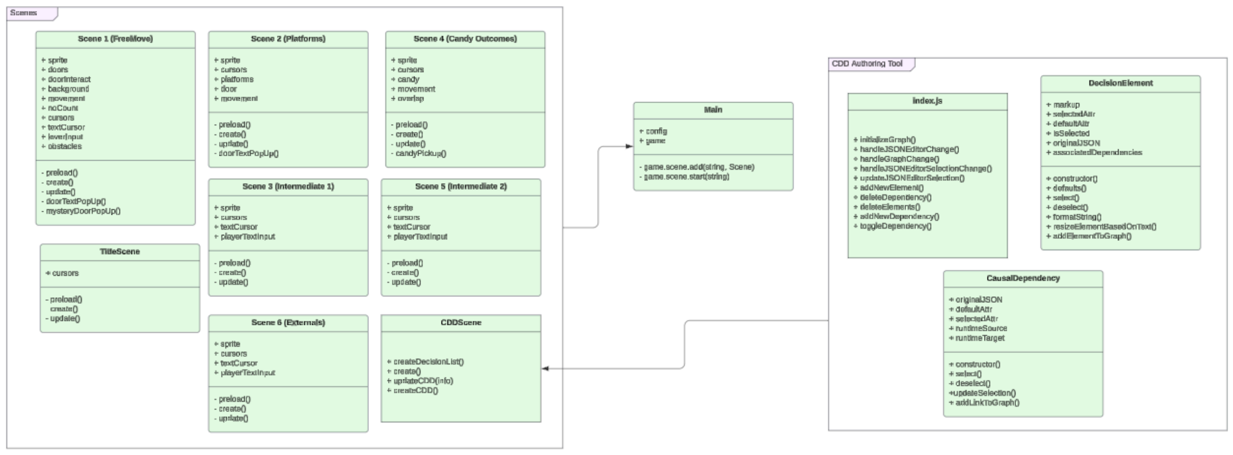
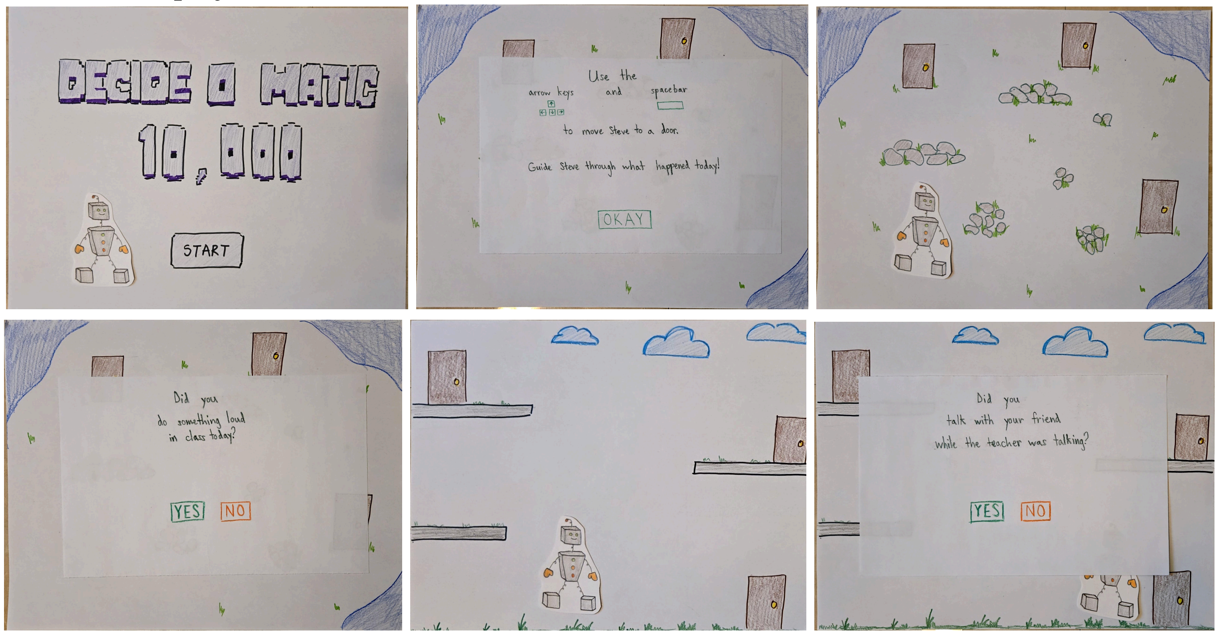


Figure 2. UML Class Diagram

GUI Design

Paper Prototype

Unique to our project, our Katabasis sponsors required us to create a paper prototype to represent our vision of the game before coding the project. This was an early stage design requirement that happened before we started implementing our code portion. Similar to wireframes, the paper prototype goes through each scene step-by-step to simulate what an average gameplay experience should look like. After the title screen, players will see instructions for the first scene which is a basic movement scene with obstacles and must enter one of the presented doors. The following scene is a platforming scene with the same objective as the previous one. Then, students will be questioned about why they made the choice they did, how it made them feel and the results of how it made others around them feel. These last scenes have been heavily modified since designing the paper prototype but many of the original concepts have been incorporated into the coded project.


DECIDE O MATIC
10,000
START
Use the arrow keys and spacebar to move Steve to a door.
Guide Steve through what happened today!
OKAY
Did you do something loud in class today?
YES
NO
Did you talk with your friend while the teacher was talking?
YES
NO

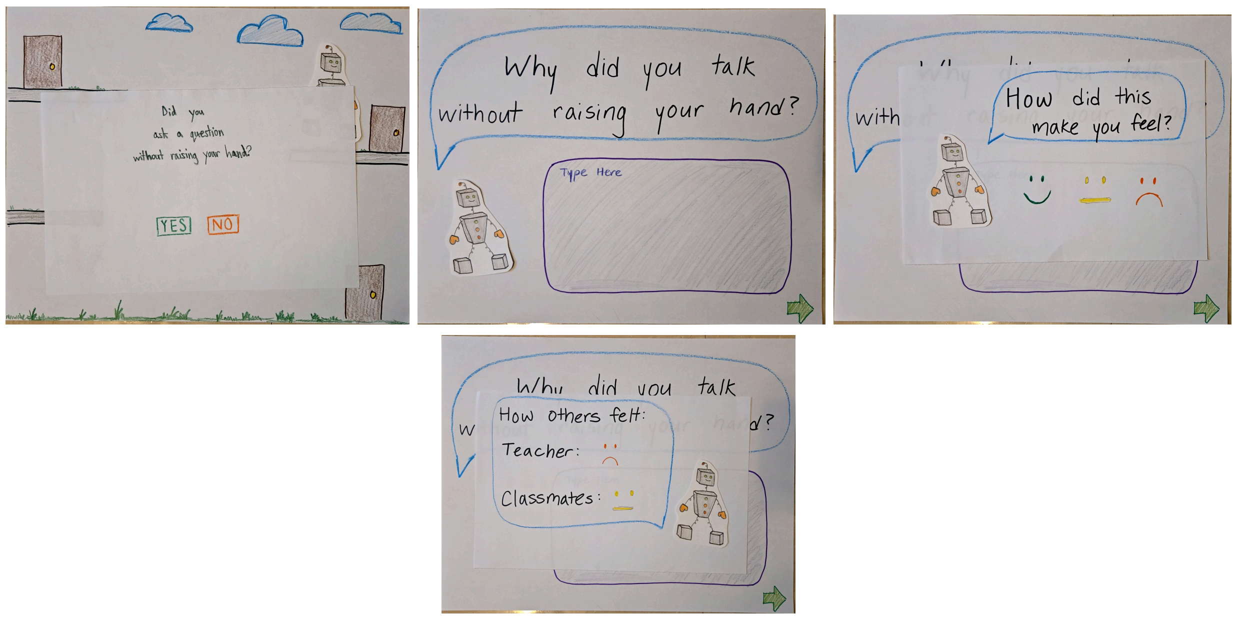

Did you ask a question without raising your hand?
YES
NO
Why did you talk without raising your hand?
Type Here
How did this make you feel?
How others felt:
Teacher:
Classmates:

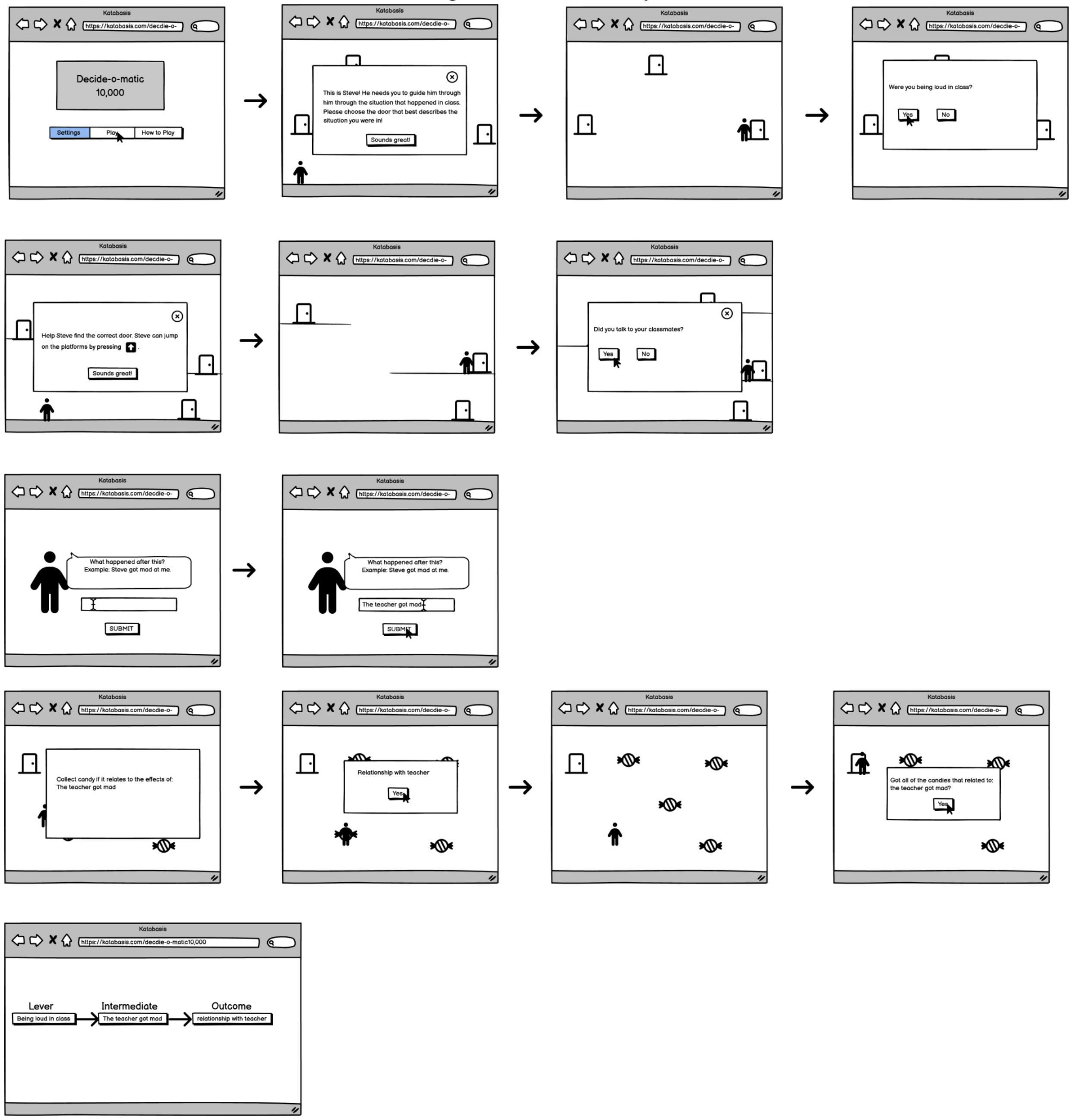
Figure 3. Early Stage Paper Prototype

Wireframes

There are two different wireframes below. The first one is the regular one, and the second one shows just the scene 1 mystery door functionality. For the regular wireframe, this describes more of an updated version of what the paper prototype is talking about. The user guides Steve to a door that describes their situation. In this case, they were being loud in class. Next, there is scene 2, which is a platformer. The user also guides Steve to a door that describes the situation again. In this case, they were talking to their classmates. Scene 3 depicts the user input scene where the user can type out what happened after. Scene 4 shows the candy scene where the user can choose outcomes that coincide with what happened. This just shows one candy being picked up however there can be more than one. As of right now, scenes 5 and 6 are not implemented. Scene 7 shows the CDD to the student.

The second wireframe shows the mystery door functionality. In scene 1, if the user does not view any of the three doors as an accurate description of what happened, the fourth door will appear. This is only after they select no for all three doors. This door will allow for the user to type out what happened in their own words. The second scene will be skipped, and then it is the same as the original design from there on.

Regular Functionality



This is Steve! He needs you to guide him through him through the situation that happened in class. Please choose the door that best describes the situation you were in!

Were you being loud in class?

Help Steve find the correct door. Steve can jump on the platforms by pressing .

Did you talk to your classmates?

What happened after this? Example: Steve got mad at me.

The teacher got mad

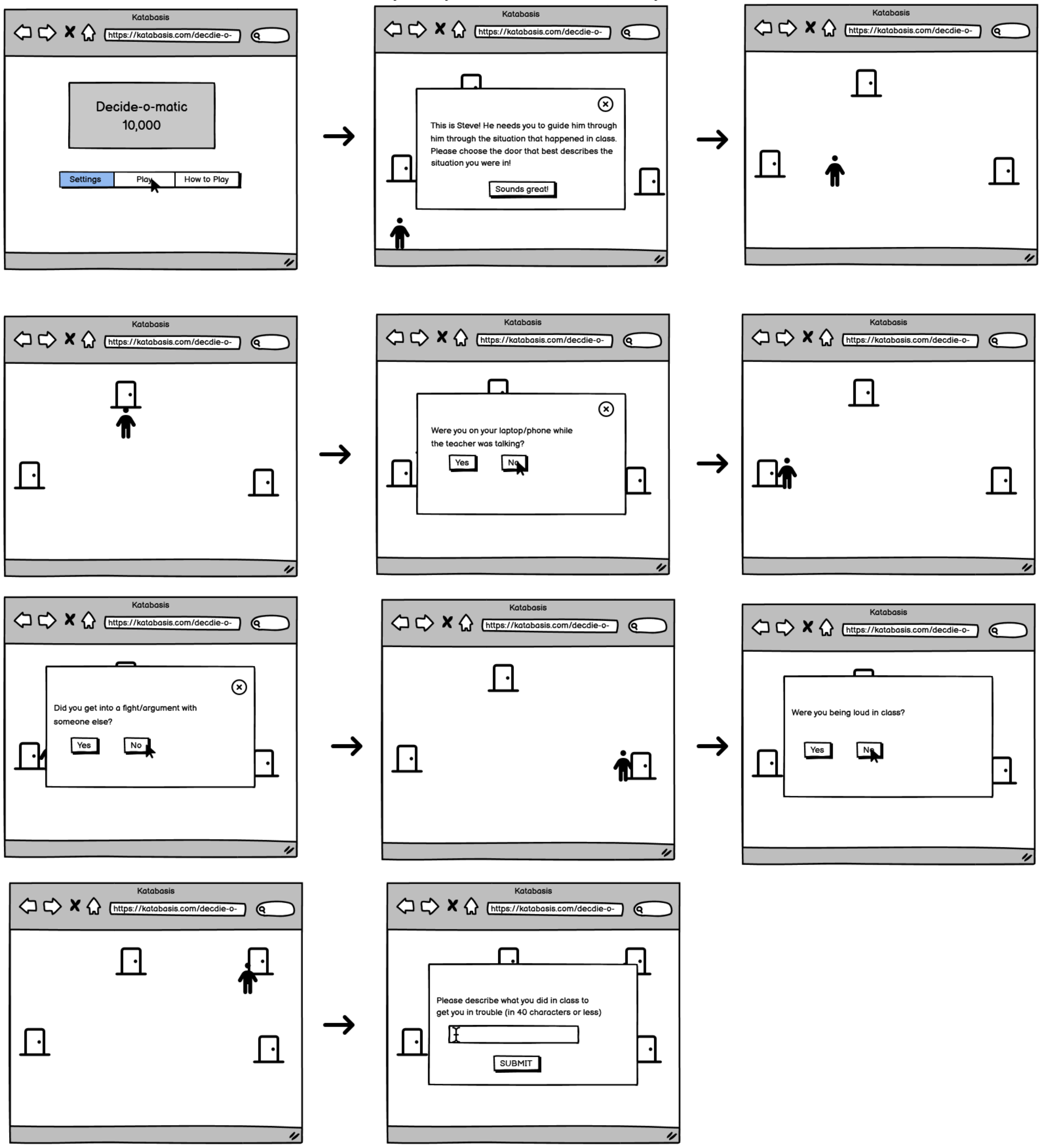
Collect candy if it relates to the effects of: The teacher got mad

Relationship with teacher

Got all of the candies that related to: the teacher got mad?

Being loud in class → The teacher got mad → relationship with teacher

Mystery Door Functionality



This is Steve! He needs you to guide him through him through the situation that happened in class. Please choose the door that best describes the situation you were in!

Were you on your laptop/phone while the teacher was talking?

Did you get into a fight/argument with someone else?

Were you being loud in class?

Please describe what you did in class to get you in trouble (in 40 characters or less)

Figure 4. Wireframes

Implementation

Author(s): Minji Kang
Reviewer(s)/Editor(s): Levi Curlee, Atiya Rulianto, Lillie Sharpe

Iteration Definition & Current Status

For our project, we are not following traditional feature based iterations and instead doing scene based iterations.

1. Iteration 0:
    a. Start: 1/10/25
    b. End: 1/27/25
    c. Scenes: For this iteration, we mostly focused on setting the groundwork for the project. We defined requirements, researched OpenDI and Phaser, and created the initial design and paper prototype.
2. Iteration 1:
    a. Start: 1/27/25
    b. End: 2/12/25
    c. Scenes: This iteration was focused on getting Phaser set up on our environments and creating the skeleton for the scenes. Lillie created custom assets for the game.
        i. Title Scene: Atiya drew a quick title page and was able to load it into Phaser.
        ii. Free Moving Scene: Steve was moving via user input through the keyboard arrows. The collision between Steve and the door was also working [GR 2].
        iii. Platform Scene: Gravity was implemented and Steve was able to jump between the platforms [GR 5].
        iv. CDD Scene: The CDD displayed in game. Player choices are not reflected, but working towards [GR8].
3. Iteration 2:
    a. Start: 2/12/25
    b. End: 3/1/25
    c. Scenes: This iteration has the bulk of our implementation for the game. The game is playable and has been usability tested by friends and family.
        i. Free Moving Scene: Began working on obstacles for Scene 1 in order to make the game more interesting [GR 3]. Door interactions with pop ups are working properly and the mystery door with text input has been created [GR 4].
        ii. Platform Scene: Doors have been added, but mystery door has not been added.
        iii. Intermediate 1 Scene: Text input for this scene has been created and it sends the data to the CDD Scene [GR 6].
        iv. Outcome Scene: Steve is able to collect the candy related to an outcome [GR 2]. Collisions between Steve and candies work properly.

        v. CDD Scene: CDD displays player choices from previous scenes [GR 8] and descriptions have been added to explain the lever, intermediate, external, and outcome [GR 10].

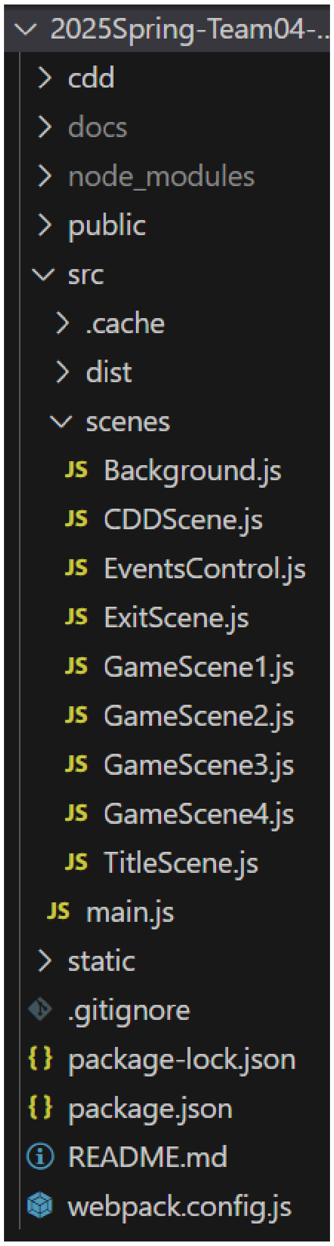
4. Current Status:
    a. Usability testing feedback has been evaluated and action steps have been created to focus on:
        i. An introduction to Steve scene
        ii. Adding a back button to the mystery door
        iii. Fixing the wording of the outcomes
        iv. Adding more instructions
        v. Creating the Intermediate 2 Scene and the External Scene
        vi. Adding a looping to start option
5. Future:
    a. Since we are doing scene based iterations, we do not have exact start and end dates planned for the iterations. Some iterations may be quicker than others depending on the issues we face during the implementation of the scenes. However, we must still implement [GR 3, 6, 10, 11, 12]. This will entail debugging obstacles, creating the scenes for Intermediate 2 and External, implementing the editing feature for the CDD, creating the ability to restart the game, and running through a good example of what could have happened in class instead. This will all be completed before April 9th for the Live Demo for Katabasis.

Security Considerations

For the Decide-o-matic 10,000, there are not very many security concerns. It does not store any information about the player and all data is volatile. It does not send any POST requests where the user could potentially manipulate the request data. It also does not have parameters in the GET requests where the user could manipulate them. There are no logins required for the game, so there are no security concerns for usernames and passwords.

In the future, if the sponsors decide to connect the game to a database, then there could be privacy concerns. If students are required to sign in, their usernames and passwords would need securing. Also, they may input some sensitive information or identifiable information into the game. Storing this information in a database always has risks of information leaks.

Project Folder Structure



```
2025Spring-Team04-...
  cdd
  docs
  node_modules
  public
  src
    .cache
    dist
    scenes
      Background.js
      CDDScene.js
      EventsControl.js
      ExitScene.js
      GameScene1.js
      GameScene2.js
      GameScene3.js
      GameScene4.js
      TitleScene.js
    main.js
  static
  .gitignore
  package-lock.json
  package.json
  README.md
  webpack.config.js
```

We are trying to follow the Phaser recommended file structure. There are four main folders: cdd, public, src, and static. The docs folder is autogenerated from webpack.config.js and node_modules stores dependencies. Both folders are in gitignore to prevent cluttering in the repository.

The cdd folder stores the code from the OpenDI CDD Authoring Tool. The public folder stores the assets for the game. The src folder stores the scenes and the main.js file. The static folder stores the index.html required for webpack to load the game.

Outside of these main folders, there is a package.json and webpack.config.js which are used for the configuration.

Project Configuration/Settings

The configuration files are the package.json and webpack.config.js files. When deploying the system on a different environment, if the node_modules folder or the docs folder is not present, the user may have to run “npm run build” in order to build the necessary dependencies. The configuration for Phaser is in the main.js file which controls the configuration for the whole game.

Testing

Author(s): Lillie Sharpe
Reviewer(s)/Editor(s): Atiya Rulianto, Minji Kang, and Levi Curlee

Overall View

Since our project is a game, we have focused primarily on two different types of testing: acceptance testing and usability testing.

While unit testing is generally used in different types of programs to ensure that they run correctly from a white box perspective, using unit testing for a game is difficult. For a game system, we are using Phaser as our game engine, which does not have any unit testing softwares that we are currently aware of. Also, much of the functionality of a game is easier to test by experiencing it through the game itself since the game would have to be run and played for different mechanics to be observable.

Because of this, we have decided to focus primarily on acceptance testing and usability testing. Acceptance testing, or system testing, is a method of testing that we can use as the programmers of our game to test features that we know should exist in our game, that we want to exist in our game, or, if we locate a bug, to keep track of the steps required to recreate a bug so that we can fix it and ensure that it is fixed. Usability testing is a type of testing where we allow playtesters who do not know how the program behind the game works to test our game. Using the feedback that we receive from usability testing, we will be able to be notified of any issues or bugs that our testers run into, any issues that our playtesters might experience with the game, or any features that playtesters might recommend to be added to the game for playability, comfort, or accessibility.

All of the testing that we have performed thus far for our game has been through computer systems, specifically laptops. All of the system tests have been performed through a Lenovo Yoga with an Intel(R) Core(TM) i5-10210U CPU processor that is x64-based, 8 gigabytes of RAM, and a 64-bit operating system. The usability tests were performed on various different models of laptop computers, including four on the Lenovo Yoga specified previously and five on other computer models.

Since the program that we are testing is a game that is built to be used on a browser, it would be best tested on a computer system. The user who would like to test our game needs to download our game from GitHub and open it through VSCode. The user must also ensure that they have npm downloaded onto their system. To run the game, execute the following two commands: "npm run build" and then "npm run start". The game should run in a separate window, and the acceptance tests and usability tests can be executed.

Unit Testing

Unit testing has been a minor roadblock for our project. For our game, we are using Phaser as our game engine. To our knowledge, Phaser does not have any designated unit testing softwares. Because our entire game is written in Phaser, we were initially uncertain about how we should go about performing unit tests. Also, using unit testing in our project would not be as useful for testing our project as it would be for most software projects since our project is a game. For most softwares, unit testing allows programmers to ensure that their code is working properly from a white box perspective. However, since our program is a game, we must run our game and interact with it to understand if our code is working properly.

As a result of unit testing not being quite as useful for our project as it would be for others and also due to our current inability to use unit testing, we have been permitted not to use unit testing for the Phaser portion of our project, which is currently the only fully functional portion of our project that exists so far. Because of this, we will instead be relying on our acceptance testing and usability testing to give us feedback on the functionality and progress of our project.

System or acceptance testing is a type of black box testing that we can actively perform on our own as the programmers of our game, knowing the contents of our own code. System testing is the process of writing out manual system tests to complete and incrementally running through them as indicated by their designated steps. If we get the expected result listed for each system test, then the test is passing, and if not, the test is failing. By using system testing, we can test the functionality of features that we know already exist in our code or that we are currently implementing in our code. We can also use system testing to outline new features and functionality that we want to exist in our game. Last but not least, system testing can also give us a useful method of keeping track of bugs and issues that need to be fixed. By running through system tests that relate to these bugs, we know whether our attempts to fix them are successful or not.

Usability testing is similar to system testing in that it is also a type of black box testing, but for this type of testing, we invite testers who do not know the inner workings of our program to play our game. Since they do not know how our game software works, they will be able to find issues with our program that we would not otherwise find. Through their playtesting, we will receive feedback related to the playability and accessibility of our game, and they will likely make recommendations for our existing features to be improved, for new features to be added, and for current features to be removed or changed.

Acceptance Testing

The following table includes all of our acceptance or system tests, of which there are currently thirty-two (32). Fourteen out of thirty-two (14/32) of our system tests, starting at test [19], were completed through our "obstacles" branch, not our "development" branch. The rest of our tests were completed on our "development" branch. To go through all of these tests as we have, please checkout the "development" or "obstacles" branch based on which test is to be tested, as I have detailed above.

Test	**Description**	**Expected Results**	**Actual Results**
[1] GameStart	1. Initialize the game as indicated above. 2. Press the spacebar to start the game.	The game should start. The player should see an instruction pop-up.	The game does start, and an instruction pop-up does appear. The pop-up has an “Okay” button. This test passes.
[2] InstructionsPopUp	Preconditions: [1] passes and game is running. 1. Click the “Okay” button.	The player should see a maze-like level (Scene 1).	A maze-like level does appear after the “Okay” button is pressed. This test passes.
[3] MovementMazeRight	Preconditions: [1] and [2] pass and game is running. 1. Press the right arrow key.	The player character should move right.	The player character does move right when the right arrow key is pressed. This test passes.
[4] MovementMazeLeft	Preconditions: [1] and [2] pass and game is running. 1. Press the left arrow key.	The player character should move left.	The player character does move left when the left arrow key is pressed. This test passes.
[5] MovementMazeUp	Preconditions: [1] and [2] pass and game is running. 1. Press the up arrow key.	The player character should move up.	The player character does move up when the up arrow key is pressed. This test passes.
[6] MovementMazeDown	Preconditions: [1] and [2] pass and game is running. 1. Press the down arrow key.	The player character should move down.	The player character does move down when the down arrow key is pressed. This test passes.
[7] PopUpScene1DoorA	Preconditions: [1], [2], [3], and [6] pass and game is running. 1. Move the player character to the door to the far right of the player character after initial spawn using the right and down arrow keys.	A pop-up should appear that states, “Were you being loud in class?” The pop-up should have “Yes” and “No” buttons.	A pop-up that states, “Were you being loud in class?” does appear. It does have a “Yes” button and a “No” button. This test passes.

[8] PopUpScene1DoorB	Preconditions: [1], [2], [3], and [5] pass and game is running. 1. Move the player character to the door above the player character after initial spawn using the right and up arrow keys.	A pop-up should appear that states, “Were you on your laptop/phone while the teacher was talking?” The pop-up should have “Yes” and “No” buttons.	A pop-up that states, “Were you on your laptop/phone while the teacher was talking?” does appear. The pop-up does have a “Yes” button and a “No” button. This test passes.
[9] PopUpScene1DoorC	Preconditions: [1], [2], [4], and [6] pass and game is running. 1. Move the player character to the door to the left of the player character after initial spawn using the left and down arrow keys.	A pop-up should appear that states, “Did you get into a fight/argument with someone else?” The pop-up should have “Yes” and “No” buttons.	A pop-up that states, “Did you get into a fight/argument with someone else?” does appear. The pop-up does have a “Yes” button and a no button. This test passes.
[10] SceneChangeScene1DoorA	Preconditions: [1], [2], [3], [6], and [7] pass and game is running. 1. Select the “Yes” button in the pop-up that appears.	The scene should change from a maze map (Scene 1) to a platformer map (Scene 2A).	The scene does change from a maze to a platformer map. This test passes.
[11] NoScene1DoorA	Preconditions: [1], [2], [3], [6], and [7] pass and game is running. 1. Select the “No” button in the pop-up that appears.	The player should remain on the maze map that they were previously playing on (Scene 1).	The player does remain on the maze map that they were previously playing on. They are also teleported away from the door. This test passes.
[12] SceneChangeScene1DoorB	Preconditions: [1], [2], [3], [5], and [8] pass and game is running. 1. Select the “Yes” button in the pop-up that appears.	The scene should change from a maze map (Scene 1) to a platformer map (Scene 2B).	The scene does change from a maze to a platformer map. This test passes.
[13] NoScene1DoorB	Preconditions: [1], [2], [3], [5], and [8] pass and game is running. 1. Select the “No”	The player should remain on the maze map that they were previously playing on	The player does remain on the maze map that they were previously playing on. They are

	button in the pop-up that appears.	(Scene 1).	also teleported away from the door. This test passes.
[14] SceneChangeScene1DoorC	Preconditions: [1], [2], [4], [6], and [9] pass and game is running. 1. Select the “Yes” button in the pop-up that appears.	The scene should change from a maze map (Scene 1) to a platformer map (Scene 2C).	The scene does change from a maze to a platformer map. This test passes.
[15] NoScene1DoorC	Preconditions: [1], [2], [4], [6], and [9] pass and game is running. 1. Select the “No” button in the pop-up that appears.	The player should remain on the maze map that they were previously playing on (Scene 1).	The player does remain on the maze map that they were previously playing on. They are also teleported away from the door. This test passes.
[16] MovementPlatformJump	Preconditions: [1], [2], [3], [6], [7], and [10] pass and game is running. 1. Press the up arrow key.	The player character should jump up, and gravity physics should bring the player character back to a surface.	The player character jumps up, collides with the platform above it, then descends back to the platform that it started from due to gravity physics. This test passes.
[17] MovementPlatformRight	Preconditions: [1], [2], [3], [6], [7], and [10] pass and game is running. 1. Press the right arrow key.	The player character should move right.	The player character does move to the right when the right arrow key is pressed. This test passes.
[18] MovementPlatformLeft	Preconditions: [1], [2], [3], [6], [7], and [10] pass and game is running. 1. Press the left arrow key.	The player character should move left.	The player character does move to the left when the left arrow key is pressed. This test passes.
[19] ObstacleLeftCollisionRight	Preconditions: [1], [2], [3], [4], [5], and [6] pass and game is running. 1. Press and hold the left arrow key.	The player character should collide with the leftmost obstacle on its right side and be unable to move through it.	Tested through the “obstacles” branch. The player does collide with the leftmost obstacle from its right side. This test passes.

[20] ObstacleLeftCollisionBelow	Preconditions: [1], [2], [3], [4], [5], and [6] pass and game is running. 1. Use the arrow keys to move the player character beneath the leftmost obstacle. 2. Press and hold the up arrow key.	The player character should collide with the leftmost obstacle on its bottom side and be unable to move through it.	Tested through the “obstacles” branch. The player does collide with the leftmost obstacle from beneath. This test passes.
[21] ObstacleAboveCollisionRight	Preconditions: [1], [2], [3], [4], [5], and [6] pass and game is running. 1. Use the arrow keys to move the player character to the right of the topmost obstacle. 2. Press and hold the left arrow key.	The player character should collide with the topmost obstacle on its right side and be unable to move through it.	Tested through the “obstacles” branch. The player does collide with the topmost obstacle from its right side. This test passes.
[22] ObstacleAboveCollisionBelow	Preconditions: [1], [2], [3], [4], [5], and [6] pass and game is running. 1. Use the arrow keys to move the player character below the topmost obstacle. 2. Press and hold the up arrow key.	The player character should collide with the topmost obstacle on its bottom side and be unable to move through it.	Tested through the “obstacles” branch. The player does collide with the topmost obstacle from beneath. This test passes.
[23] ObstacleBelowCollisionLeft	Preconditions: [1], [2], [3], [4], [5], and [6] pass and game is running. 1. Use the arrow keys to move the player character to the left of the bottommost obstacle. 2. Press and hold the right arrow key.	The player character should collide with the bottommost obstacle on its left side and be unable to move through it.	Tested through the “obstacles” branch. The player does collide with the bottommost obstacle from its left side. This test passes.
[24] ObstacleBelowCollisionAbove	Preconditions: [1], [2], [3], [4], [5], and [6] pass and game is	The player character should collide with the bottommost obstacle	Tested through the “obstacles” branch.

	running. 1. Use the arrow keys to move the player character above the bottommost obstacle. 2. Press and hold the down arrow key.	from above and be unable to move through it.	The player does collide with the bottommost obstacle from above. This test passes.
[25] ObstacleBelowCollisionRight	Preconditions: [1], [2], [3], [4], [5], and [6] pass and game is running. 1. Use the arrow keys to move the player character to the right of the bottommost obstacle. 2. Press and hold the left arrow key.	The player character should collide with the bottommost obstacle on its right side and be unable to move through it.	Tested through the “obstacles” branch. The player does collide with the bottommost obstacle from its right side. This test passes.
[26] ObstacleRightCollisionLeft	Preconditions: [1], [2], [3], [4], [5], and [6] pass and game is running. 1. Use the arrow keys to move the player character to the left of the rightmost obstacle. 2. Press and hold the right arrow key.	The player character should collide with the rightmost obstacle on its left side and be unable to move through it.	Tested through the “obstacles” branch. The player does collide with the rightmost obstacle from its left side. This test passes.
[27] ObstacleRightCollisionAbove	Preconditions: [1], [2], [3], [4], [5], and [6] pass and game is running. 1. Use the arrow keys to move the player character above the rightmost obstacle. 2. Press and hold the down arrow key.	The player character should collide with the rightmost obstacle from above and be unable to move through it.	Tested through the “obstacles” branch. The player does collide with the rightmost obstacle from above. This test passes.
[28] ObstacleRightCollisionRight	Preconditions: [1], [2], [3], [4], [5], and [6] pass and game is running. 1. Use the arrow keys	The player character should collide with the rightmost obstacle on its right side and be unable to move through	Tested through the “obstacles” branch. The player does collide with the rightmost

	to move the player character to the right of the rightmost obstacle. 2. Press and hold the left arrow key.	it.	obstacle from its right side. This test passes.
[29] ObstacleLeftCollisionRightMove	Preconditions: [1], [2], [3], [4], [5], and [6] pass and game is running. 1. Use the arrow keys to move the player character to the right of the leftmost obstacle. 2. Press and hold the left arrow key. 3. Press any other arrow key (up, down, or right).	The player character should collide with the leftmost obstacle on its right side and should not pass through it, even after the player presses another movement button.	Tested through the “obstacles” branch. Three different tests were run with these instructions using each of the different directional keys mentioned. Pressing the up arrow key after collision was successful. Pressing the down arrow key after collision was unsuccessful. Pressing the right arrow key after collision was also unsuccessful. The unsuccessful attempts caused an abnormal rapid movement in the opposite direction from the direction where movement was intended to go in, even causing the player to pass through the obstacle. This test fails.
[30] ObstacleAboveCollisionBelowMove	Preconditions: [1], [2], [3], [4], [5], and [6] pass and game is running. 1. Use the arrow keys to move the player character below the topmost obstacle. 2. Press and hold the	The player character should collide with the topmost obstacle on its bottom side and should not pass through it, even after the player presses another movement button.	Tested through the “obstacles” branch. Three different tests were run with these instructions using each of the different directional keys mentioned.

	up arrow key. 3. Press any other arrow key (left, down, or right).		Pressing the left arrow key after collision was successful. Pressing the down arrow key after collision was successful. Pressing the right arrow key after collision was unsuccessful. The unsuccessful attempt caused an abnormal rapid movement in the opposite direction from the direction where movement was intended to go in, even causing the player to pass through the obstacle. This test fails.
[31] ObstacleBelowCollisionAboveMove	Preconditions: [1], [2], [3], [4], [5], and [6] pass and game is running. 1. Use the arrow keys to move the player character above the bottommost obstacle. 2. Press and hold the down arrow key. 3. Press any other arrow key (left, up, or right).	The player character should collide with the bottommost obstacle on its top side and should not pass through it, even after the player presses another movement button.	Tested through the "obstacles" branch. Three different tests were run with these instructions using each of the different directional keys mentioned. Pressing the left arrow key after collision was successful. Pressing the up arrow key after collision was successful. Pressing the right arrow key after collision was unsuccessful. The unsuccessful attempt caused an abnormal rapid movement in the opposite direction from

			the direction where movement was intended to go in, even causing the player to pass through the obstacle. This test fails.
[32] ObstacleRightCollisionLeftMove	Preconditions: [1], [2], [3], [4], [5], and [6] pass and game is running. 4. Use the arrow keys to move the player character to the left of the rightmost obstacle. 5. Press and hold the right arrow key. 6. Press any other arrow key (left, down, or up).	The player character should collide with the rightmost obstacle on its left side and should not pass through it, even after the player presses another movement button.	Tested through the "obstacles" branch. Three different tests were run with these instructions using each of the different directional keys mentioned. Pressing the left arrow key after collision was unsuccessful. Pressing the down arrow key was successful. Pressing the up arrow key was successful. The unsuccessful attempt caused an abnormal rapid movement in the opposite direction from the direction where movement was intended to go in, even causing the player to pass through the obstacle. This test fails.

Overall, considering the acceptance tests that we currently have written and completed, I believe that our current system tests tell us that our project is not fully complete or correct, but we are making great progress and focusing on ensuring that our features work properly once they are implemented. Because of the large number of tests that we have written and even passing, I can safely say that we are working hard and putting our best effort towards this project. Whenever there are new features that we are working on, our team does our best to incorporate them in our system tests to ensure that they are functioning correctly. Out of these system tests, our team has twenty-eight out of thirty-two (28/32) passing. All of our "development" branch tests are passing currently, and ten out of fourteen (10/14) of our "obstacles" branch tests are passing. Overall, it is incredible that we have so many system tests written and even passing at this stage of our project. Not only do we have a hefty number of system tests, but we also have a majority of them passing. The large number of our tests that are written and passing tells us that we are not only implementing a large number of features in our project, but many of these features are also already fully correct, save for the obstacles feature, which is only partially correct.

Other Testing

Since we could not perform unit testing on our project, we decided instead to utilize usability testing to give us more feedback. Through usability testing, we permitted people who knew nothing about how our game works to play our game using the below usability testing script. The results that we received from their tests gave us much needed feedback about what their overall impression of the game was, the issues that they encountered with the game, and the bugs that they might have encountered in the game. They also gave us a wealth of recommendations regarding features that would improve playability, understanding, and accessibility for students who would play our game in the future.

To perform our usability testing, we each accessed the game from our own laptop computer devices. We each have our game downloaded from GitHub on VSCode. We each also have npm installed on our devices. From VSCode, we type in the commands "npm run build" then "npm run start" to start our game in a browser window. Then, we ran through the below usability testing script with our tester.

Below, in italics, is the usability testing script that we used commonly throughout all of our unit tests:

Usability Testing Script

Introduction:

- *Hello! Thank you for participating in this usability test for the Decide-O-Matic 10,000.*
    - *Introduce yourself and team.*
- *The Decide-O-Matic 10,000 is a game designed for middle school students that get sent out of class due to their emotional or behavioral disorders. It helps them understand the connections between their actions and the outcomes.*
- *We are trying to gather information on how users interact with the current version of the game.*
- *There are no right or wrong answers, and we encourage you to think out loud during this process.*
- *Could you tell us your name, age, and profession?*

Tasks:

1. *Imagine you are a middle school student who just got sent out of class for your behavior.*
    1. *Please tell us what you did.*

2. *The teacher hands you this game and your task is to complete the game.*
3. *Please start the game.*
4. *Walk us through your thought process for each step.*

Wrapping Up:

- *What was your overall impression of the game?*
- *What specific issues did you have during the game?*
- *Are there any features that we could add to improve the experience?*
- *Thank you so much for your time and please reach out to us if you have any additional questions, comments, or concerns.*

We completed nine usability tests, which were all fruitful with information. Then, as a team, we compiled the results that we received from each usability test into a document so that we could all understand them easier, the contents of which is shown below.

Below, in italics, are the contents of the document that contains all the results from our nine usability tests:

Overall:

- *Game was cute*

Common Trends / Issues:

- *Candy outcome scene was unclear*
- *Keep Yes/No consistent*
- *Outcomes in general not very clear until the end*
- *Some characters not allowed in text input and the character limit*
- *Characters overflow text box*
- *Move away from mouse controls*

Recommendations:

- *Use WASD and arrow keys to move Steve (or just WASD)*
- *Change up arrow to spacebar to jump or add instructions*
- *Some form of abusive language detection*
  - *Like abuse or trauma in the classroom/home*
- *Adding question mark to reintroduce instructions in case they forgot*
- *Add input , ‘ and numbers for text input*
- *Making “yes” and “no” buttons instead of just words*
- *Decrease Steve size for candy outcome scene*
- *Candy outcome scene question box is too big, cannot see Steve*
- *Decrease jump in platforms scene / fall faster*
- *Back button or are you sure (ask again)*
- *For outcomes, good / bad button instead of yes / no*

    - *Ex. relationship with teacher (good/bad)*
    - *Ex. level of trouble after class (small/medium/large)*
- *Candies were too general in description*
    - *Use of word "level" can be confusing due to correlation to numbers*
- *Allow for student to put multiple levers/actions*
- *To select "Okay" or "Submit", use spacebar or enter*
- *For "Yes" and "No" buttons*
    - *Suggestion 1: map to arrow keys*
    - *Suggestion 2: create a cursor that maps to arrow keys instead of mouse*
- *Add essay section to allow students to decompress*
    - *Spectator opinion: finished CDD section may accomplish this*
- *Show completed diagram at the end after finish button*
    - *Or show it completely at start and then run through it*
- *Create delay to avoid players spamming through CDD Scene*

Takeaways:

- *Change outcomes*
- *Add more instructions*
- *Clean up UI*

We received a large amount of useful information in the form of the overall picture perceived, issues encountered, and recommendations given by our usability testers in their feedback. Overall, our main takeaways from the feedback that we received includes changing the contents of the outcomes and how they are presented, adding more instructions for user understanding, and cleaning up our user interface (UI).

In our usability test feedback, we received a general consensus that the outcomes stage of our game needs an overhaul. Many testers had issues with that part of the game because the wording of the pop-ups that they received from interacting with the candy objects was too confusing to understand, and they were unsure how to say no to collecting the candy objects as opposed to entering the doors in other stages of the game.

We also need to add more instructions to our game and clean up our user interface. Several testers indicated that they were having difficulties with the controls presented by the game, such as player movement or mouse interactions, because they were not clearly explained on different levels. If we change our instructions for each stage to include their respective controls, then our players would no longer be uncertain which controls to use in each stage. On the other hand, if we also clean up our user interface by making how players perform certain actions, such as selecting "Yes" and "No" buttons, more consistent instead of changing their mechanics between levels, then we would be able to minimize the amount of new instructions that we would need to add.

Task Plan

Author(s): Minji Kang
Reviewer(s)/Editor(s): Levi Curlee, Atiya Rulianto, Lillie Sharpe

Team 4 Katabasis 1 Task Plan			
Item	Owner(s)	Due Date	Status
Sign IP Agreement	ALL	1/17/25	Complete
Design:			
● Low level design	MK, AR	1/20/25	Complete
● Wireframes	LC		Complete
● Paper prototype	LS, MK		Complete
● Requirements	ALL		Complete
● Choose game engine	ALL		Complete
OPR 1 Slides	ALL	2/4/25	Complete
OPR 1 Presentation	LS	2/5/25	Complete
Implementation:			
● Title Scene	AR	2/12/25	Complete
● Scene 1 (Free Moving)	AR		Complete
● Scene 2 (Platform)	LC		Complete
● CDD display	MK		Complete
● Mystery Door	AR	3/1/25	Complete
● Text Input	AR, LC		Complete
● Intermediate 1 Scene	LC		Complete
● Outcome Scene	LC		Complete
● Player choices impacting CDD	MK		Complete
● Obstacles	LS, AR		Complete

Testing:			
● Usability Testing	ALL	3/2/25	Complete
OPR 2 Slides	ALL	3/4/25	Complete
OPR 2 Presentation	LC, MK	3/5/25	Complete
Implementation:			
● Intro to Steve Scenes	MK	3/26/25	Completed
● Back button on mystery door	AR		Completed
● More instructions	ALL		In progress
● Fix wording of outcomes	ALL		Completed
● Make CDD more understandable	MK		In progress
● Intermediate 2 Scene	LC		Completed
● External Scene	LC		Completed
● Artwork Updates and Waving Animations	LS		Completed
● Title Screen	LS	4/2/25	In progress
● Loop to start	ALL		Not Started
● Walk through what could have happened instead (good example)	ALL		Not Started
Posters & Pies	ALL	4/23/25	In progress