Quine Final Report

Vannie Kopalakrishnan

Introduction

The aim of this paper is to understand what design features make an educational videogame an effective tool for learning in the classroom. This was done by conducting embedded design research; by developing a game and redesigning elements of the game after playtests and feedback, I could see what and how the implementation (or lack of) certain design features made my game more or less effective as an educational tool.

Game Description

Quine is a 2D educational videogame that teaches users about basic programming logic. This is done by allowing players to solve programming puzzles as the main task for each level. As the levels progress, the programs become more challenging, requiring the user to apply what they have learned in previously solved puzzles, to current levels. The target audience for this game are students with little to no programming experience. Hence, users are introduced to variable assignments, building and reusing functions, if statements, Boolean logic, and logical operations in this game.

Game Mechanics

The main game mechanic for Quine is the use of a ‘drag and drop’ system to solve basic programming puzzles. The player can control a robot and interact with items in the level using the keyboard. As the player interacts with items such as a computer or a key, a computer interface will appear, allowing the player to read a program and fill in missing blanks with coding blocks that illustrate programming logic ideas such as Boolean logic. The main objective of each level is to unlock the exit by solving these puzzles. Levels must be completed in succession, making the game progression linear. This is done in order to reinforce past concepts introduced at starting levels.

Design Process

The original design of Quine included a story-telling mechanic where the main motivation of each level was to discover clues over what happened to the ship that you, as a robot, are on. This was to be done by “hacking” the ship’s computer networks which was to be done by solving programming puzzles by dragging blocks that contained pieces of code onto the computer interface. However, in the earlier weeks, it became apparent that the plot of the game was becoming the center of the game’s objective instead of the educational aspect of learning programming logic. As a result, the story-telling mechanic was scrapped.

With the story-telling mechanic gone, the next thing to do was brainstorm ideas to make the gameplay more engaging and creative. Some sources of inspiration were *RoboRally* and *Human Resource Machine*. In both games, players were able to control their character by implementing blocks of code. Similar to this, the main gameplay of Quine would change so that the player would be able to program a robot to move around a level. What was originally proposed was that players would have access to blocks that they could rearrange to change the robot’s movements.

Image (1) Original design of interactable blocks. ‘R’ would move the player right, ‘U’ up, ‘L’ left, and ‘D’ down.

This would teach players how to think like a programmer more so than teach them about specific programming concepts. This proved to be a bit of a challenge; while players were still actively learning, how the game was designed did not seem convincing enough to make players aware of what they were learning. Furthermore, implementing the game mechanic of having the player control a robot using blocks proved to be very difficult. As a result, this idea was scrapped.

Going back to the idea of teaching users about key programming concepts such as Boolean logic and logical operations, the final redesign of the game’s prototype focused more on what the player’s character could do. It was decided that the main ability of the character (other than movement) was the ability to “hack” objects. Users would be able to move a robot around that would interact and hack objects around it. By “hacking” these objects, the player would be able to unlock the exit to the next level. This idea proved to be more effective; by focusing on the idea of reprogramming an object in the level, it would be clear that the objective of the game was to solve a programming puzzle rather than uncover the plot of a story or moving a robot.

Original Findings

For the first playtesting session, 11 players were surveyed. Users were asked to fill out 3 surveys during the session:

(1) Demographic survey
The demographic survey was conducted to gather the player's age, occupation/study, and previous gaming experience.

(2) Pre-game survey
The pre-game survey was conducted in order to gather the player's experiencing with programming, their interest in the subject, their expectations for the game, and whether they would rather have a game as a substitute or supplement for learning in the classroom setting.

(3) Post-game survey
The post-game survey encompassed all the questions of the pre-game survey, but focused on how they felt after the game. The idea was to see if there was a change in their responses. Furthermore, questions about whether the game met the player's expectations, how difficult the game was, whether they would recommend the game or play it again, how much they enjoyed it, and general feedback were also asked.

Only levels highlighted with blue in Appendix A were available for the first playtest. For the first playtesting session, users had access to two levels.

Learning Outcomes

With a majority of the playtesters (63.9%) having no experience with computer science or programming, it was important to measure their perceived knowledge after playing the game. To measure how successful the learning outcomes for Quine was, I asked players how much they learned from game with a scale of 1 being nothing, and a scale of 7 being a lot. Overwhelmingly, 81.9% of playtesters found that they had learned little to none from the game. No one found that they learned a significant amount. To figure out why the game was not effective in terms of learning, I looked into the difficulty of the game. 54.6% found the game to be difficult to very challenging, while 9.1% found the game to not be challenging. With a majority finding the game to be challenging, it was important to seek feedback into why players found it difficult. When asked, many playtesters attributed their lack of understand to the lack of levels, instructions, and difficulty in puzzles to their confusion. They also felt they did not have a clear understanding of the learning outcomes for Quine. Although they were aware that the game's goal was to teach users about programming logic, they felt that levels did not illustrate this. There was a lack of awareness over what was being taught; many suggested more levels would be appropriate. It is clear that because proper learning goals and level objectives were not organized prior to developing the game, the two levels lacked a clear motive.

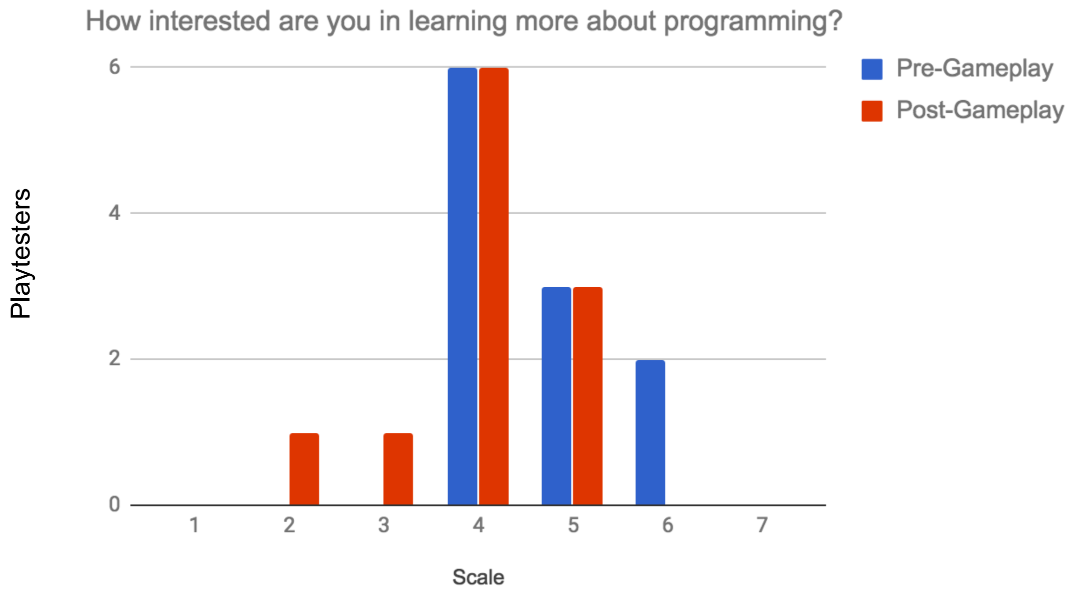
Motivation and Interest

Because my game only had two levels, it was difficult to get an accurate depiction or how motivated the playtesters were to continue playing Quine. The game was too short to draw any reliable data. However, in order to see how effective my game was, I still wanted a way to record how each playtester's interest in programming or motivation to play a game changed before playing Quine and after playing Quine. This was mainly measured using the following questions:

(1) How interested are you in learning more about programming? (Measured pre-gameplay and post-gameplay)
(2) Would you recommend this game to your friends?
(3) How much did you feel you learned about programming?
(4) Did this game meet or fail your expectations? How so?

The first three questions were measured on a scale from 1 to 7, with 1 being the least and 7 being the most. Question 4 served as a short answer response in order to understand what elements of the game needed more improvement on. I found the responses to the first question to be the most interesting because I had the ability to see whether my game had a positive or negative impact on the playtesters' interest in learning more about programming.

Graph (1) How interested are you in learning more about programming?



Treating a response of scale 4 as being neutral to the idea of learning more about programming, I established any responses with a scale higher than 4 to be a positive outlook on the subject and any response lower than 4 to be a negative outlook on the subject. My understanding was that if the responses before and after playing the game resulted in the same results for responses with a scale higher than 4, then it would be a good indicator that the game was interesting. If it yielded better results, it would be a good indicator that the game was also motivating—if users who felt their interest in programming increased after the game, it would potentially mean that the game possessed a level of influence on the users' motivation to learn more about the subject.

Overall, before the playtest approximately 45.4% of the players possessed a strong interest in the subject while 54.5% were neutral to the idea of learning to program. After the playtest, only 27.3% had a strong interest in the subject with 18.2% not interested in the subject. The percentage of users that were neutral on the topic stayed the same. Although it is important to note that there was not enough playtesters to make accurate conclusions, the figures presented served as a motivator for the redesign of Quine. With a dwindling interest, students would find themselves to be less motivated should more levels be added. Furthermore, if the interest in the subject decreased from pre-gameplay to post-gameplay, then it was clear that the game was not effective in acting as an educational tool for students' learning in the classroom setting.

Overall Game Effectiveness

From the data, it was clear that most of the playtesters found the game to be too challenging. Interestingly, many of those surveyed did have experience with programming. When asked what was difficult about the game, testers felt that the lack of instructions limited them from being able to play the game without any guidance. Furthermore, playtesters that were new or inexperienced with programming found the puzzles to be too challenging. Introducing users to loops before a simpler concept such as variable assignment left users more confused than enlightened. The combination of hard puzzles and lack of instructions made the game ineffective. This was clearly shown in the data: 81.8% of playtesters felt they did not learn anything significant after playing Quine while 72.7% would not play the game as a supplement for their learning. Furthermore, when asked whether the game met their expectations, playtesters often responded negatively to the question, stating that the game's goal (to teach users the fundamentals of programming logic) were not present in the gameplay.

Redesign

Redesigning Quine included looking into what design features were most crucial in dealing with. This was done by collecting general feedback from previous playtesters; with this feedback, I was able to prioritize what design features needed improvement the most.

Design features (ordered from most important to least important) and implementations are illustrated in the following table:

Change	Intended Design Impact	Reason of Change	Summary of Design Implementations
Develop more puzzles to reinforce educational concepts. Make the puzzles the key feature for each level.	Subject integration	With only 2 levels, there was a lack of programming concepts available in the game.	Adding more levels and making the programming puzzles meet my new learning objectives helped with subject integration.

Develop learning goals for each level.	Learning goals	Most playtesters remarked that they were not sure of what they were being taught. There was not a distinct learning plan – levels differed greatly in terms of what was being taught.	Advice was given to visit high school or first year computer science curriculum. This helped immensely. By looking at various curriculums, I was able to design my own curriculum for Quine. This helped to organize the game more, and come up with proper learning objectives for each level.
Allow players to interact with items that can link back to the terminal window.	Immersion	There was a lack of connection between moving the character around, and solving the puzzles. Players felt like they were playing two separate games instead of one.	I added two main items to each level: a computer, and a key. I also made sure that each programming puzzle incorporated these items. For example, players can interact with the computer to activate the puzzle, and then collect the key and “hack” the key to unlock the exit.
Add more levels.	Levels	With only 2 levels, users felt the game was too short.	More levels were added. Before the final playtest, there were 8 levels in total.
Decrease the difficulty of earlier levels.	Challenge	While and for loops were introduced in the first two levels. For those unfamiliar with programming, this made the game either too difficult or left users confused over what they were learning.	With specific learning goals in mind, I was able to decrease the difficulty of earlier levels. Earlier levels now deal with variable assignments and Boolean logic with newer levels reinforcing and adding to these concepts.
Add instructions so players know what to do.	Visual instructions	I often had to guide playtesters throughout the game. It was not very clear over what	I took out the NPC just because a lot of playtesters were not aware that they could talk to the scientist. Instead, I added a little

		the user was supposed to do in the game.	dialogue box with instructions on how to proceed for the specific level.

It is important to note that many of the proposed design feature changes are interlinked. For example, by coming up with specific learning goals in mind, I was able to redesign the levels so that the beginning was easy for users to grasp, with mid-levels focusing on reinforcing concepts introduced in earlier levels. By the end of level 8, users would have an idea of how to do variable assignments, build and reuse functions, and how programming logic works.

Level Redesign

When reading different classroom curriculums, I noticed that teachers often repeated concepts or used concepts previously taught in different topics. Keeping this in mind, incorporated repetition into my level redesign. By repeating concepts in newer levels, players would be given the chance to review what they have learned and also, learn to apply existing knowledge to new problems. The ability to apply their knowledge would solidify what they have learned before.

The following table illustrates the proposed learning objectives for each level. Levels highlighted in orange were not implemented.

Level	**Level Objective**
1	There is not learning objective. Level 1 is to allow the player to understand the mechanics of the game (i.e. moving the player to the exit)
2	Introduction to variable assignments.
3	Introduction to creating a function.
4	Boolean logic.
5	If statements.
6	This level is designed to reinforce the learning objectives from levels 2 to 3.
7	Use of previous functions (helper functions).
8	And/Or operations.
9	A review of if statements and Boolean logic using and/or operations.
10	Introduction to for loops. This can be done by illustrating how a variable changes value after each iteration of the for loop.
11	For loops with if statements.

12	Functions with for loops (review)
13	Boolean logic and how it relates to while loops.
14	Last function with while loop.

UI Changes

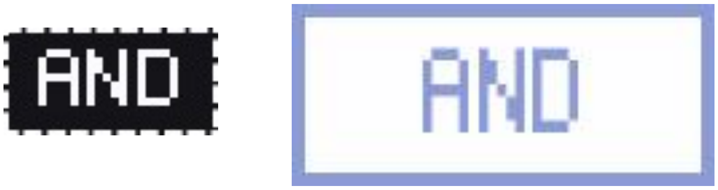
Accessibility

Some UI changes were made in order to make the game more accessible. Many players thought the screen size of the game was too small and questioned why there was not a way to pause the game. Furthermore, the game lacked a menu screen which was needed in order to introduce users to the game, and the title of the game. In the final design of the game, a menu, pause menu, and bigger screen size were all implemented.

Clarity

The computer interface within the game was completely redesigned. This included redesigning the blocks of code and added a button to reset the level. The redesign included giving the computer interface a title “Terminal Window” and keeping it separate from the score and dialogue boxes. This was done because players were often not aware that the blocks of code were objects they could interact with. They assumed they were part of the computer interface.

Image (3) Redesign of coding blocks. Left image is from the first playtest, right image is from the final playtest.



Choosing different colours, making the blocks bigger and adding borders to the blocks were done to alleviate this confusion. Furthermore, blocks that contained functions were coloured differently in order to help players differentiate between a function and an operation/variable. Significant changes are all illustrated in Appendix A.

New Findings

For the final playtesting session, 43 students were surveyed. Users were asked to fill out the same 3 surveys from the first playtest. Levels highlighted with yellow in Appendix A were available during the final playtest. Levels highlighted in blue were no longer available.

Learning Outcomes

Overall, the learning outcomes for the final playtest were significantly more successful than the learning outcomes for the first playtest. 67.4% of playtesters had either taken a computer science or programming course at school or through other resources. 32.6% of the playtesters had never taken a course in computer science or programming. However, only 21.9% had more than a year's experience with programming. This was significant since Quine was intended for students with little to no background in computer science or programming.

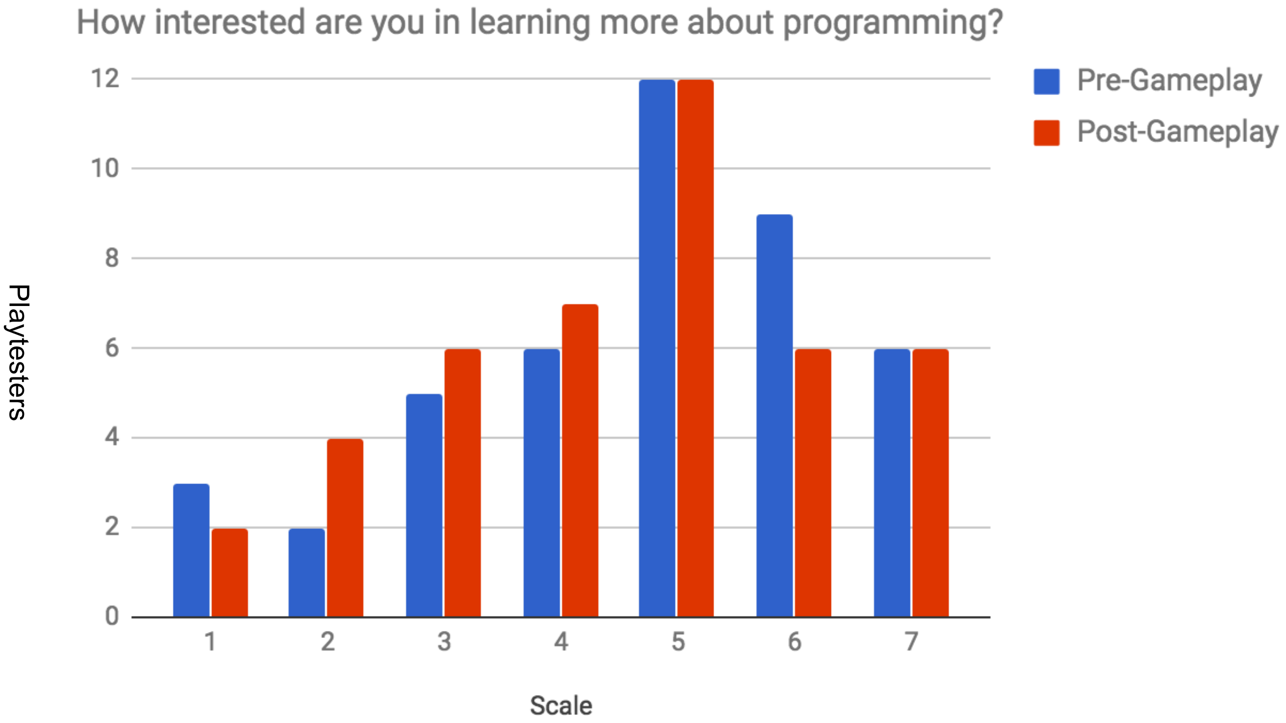
When asked how much they felt they learned after playing Quine, 25.6% learned a significant amount, while 34.7% did not learn much. Approximately 39.7% found that they had learned more about programming, but not a significant amount. This could have been attributed with the fact that many of the playtesters were previously exposed to programming and found the material within the game to be trivial or more of a review than a new concept. However, some comments in the general feedback indicated otherwise; many pointed out that users could guess the answers to the puzzles. This was because the design feature of the drag and drop system did not allow users to drag an incorrect block to a specific slot. This meant that players who felt overwhelmed or bored by the game could quickly finish the levels by guess and check instead of actively reading the code to solve the puzzles. Furthermore, there was no consequence for getting the wrong answer, giving players the incentive to guess their way through the game.

Motivation and Interest

Motivation seemed to play a big role in determining whether a user found the game enjoyable and effective in their learning. Because in the first playtest, I only had the first two levels done, I could not get a reliable understanding of whether playtesters were motivated to finish the game. However, with more levels implemented for the final playtest, I had a better understanding of the playtesters' motivation. This was largely measured through the player’s interest in the subject of programming logic before and after the game along with whether they enjoyed the game or would recommend the game to their peers. General feedback was also useful in determining what players found interesting or boring about the game.

Once again, when measuring each player’s interest in programming before and after playing Quine, I treated a response of scale 4 as being neutral to the idea of learning more about programming, and any responses with a scale higher than 4 to be a positive outlook on the subject and any response lower than 4 to be a negative outlook on the subject. The following graph depicts the results of this.

Graph (2) How interested are you in learning more about programming? (After redesigns)



Approximately 62.8% of the players possessed a strong interest in the subject while 32.1% were neutral to the idea of learning to program and 5.1% had no interest. After the playtest, 55.8% had a strong interest in the subject with 43.9% neutral and 27.9% not interested in the subject. Since a significant amount of testers were surveyed during the final playtest than the first playtest, these results are a more accurate depiction of the success of Quine’s goal. The amount of testers who held a strong interest in the subject decreased by 7% while the amount of testers who held little to no interest increased 22.8%. This seems to indicate that the game was not as effective as intended in terms of engaging and motivating students to learn about programming.

Some playtesters pointed out the redundancy of the score system. The score always resets to 100 at the start of a new level, leaving no room for players to achieve a high score or lose the game due to poor choices in previous levels. The redundancy of the score system could have attributed to the changes in perceived interest. Others pointed out that the level designs were repetitive; there was no significant change in levels to keep players interested in the game. The introduction of new items, or character abilities could have helped with this. Moreover, implementing a time limit or enemies as a consequence for losing points could have helped make the game feel more like a game (i.e. more fun).

Overall Game Effectiveness

While proper learning goals and objectives were established for each level, many playtesters still found the game to be difficult to very challenging. In fact, 46.5% expressed that they found the game difficult. When asked what they found difficult, many responded that the instructions were unclear. They were not aware of the ability to drag blocks or even interact with items. When instructions were read to relieve the confusion, many still found themselves lost, remarking that the instructions were poorly worded. This clearly, seemed to be a design flaw that limited the game's overall effectiveness.

Despite this setback, 53.5% of players felt that they would play this game as a supplement to learning in the classroom. However, only 16.3% would play the game as a complete substitution to classroom learning. The game seemed to meet some or all expectations for most playtesters; many said more level variety, elements of surprise (such as enemies) and tweaking to the score system would improve the game. Overall, the effectiveness of the game was still lacking; more design improvements would be needed to achieve clarity within the game.

Conclusion

In the initial design of Quine, playtesters were left more confused than enlightened by the idea of learning to program. There were not significant improvements in interest towards programming nor was there a positive outlook in perceived learning throughout the game. The lack of levels and design features such as visual instructions, appeared to be the main causes for this. Furthermore, the steep learning curve for players with no programming or computer science knowledge made the game too difficult and thus, ineffective.

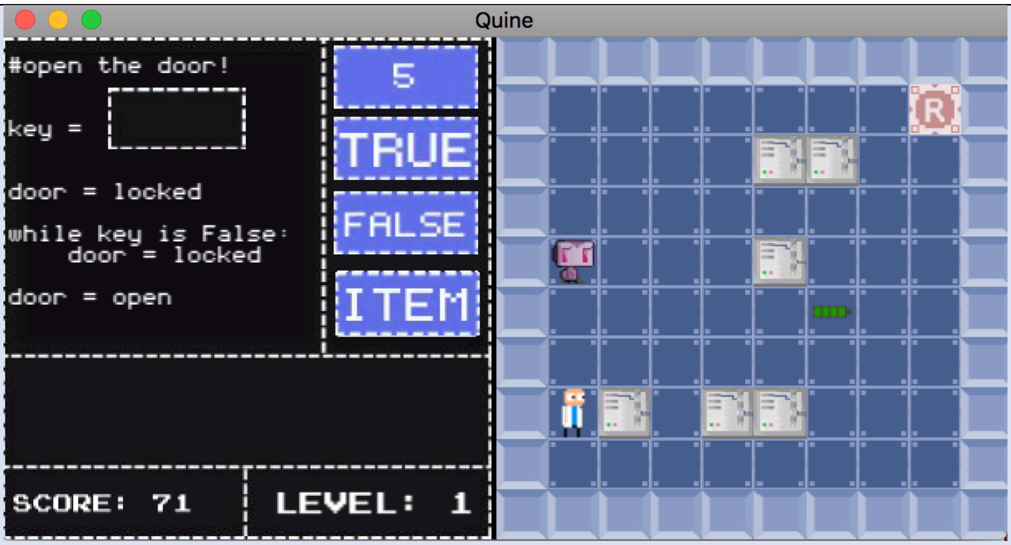
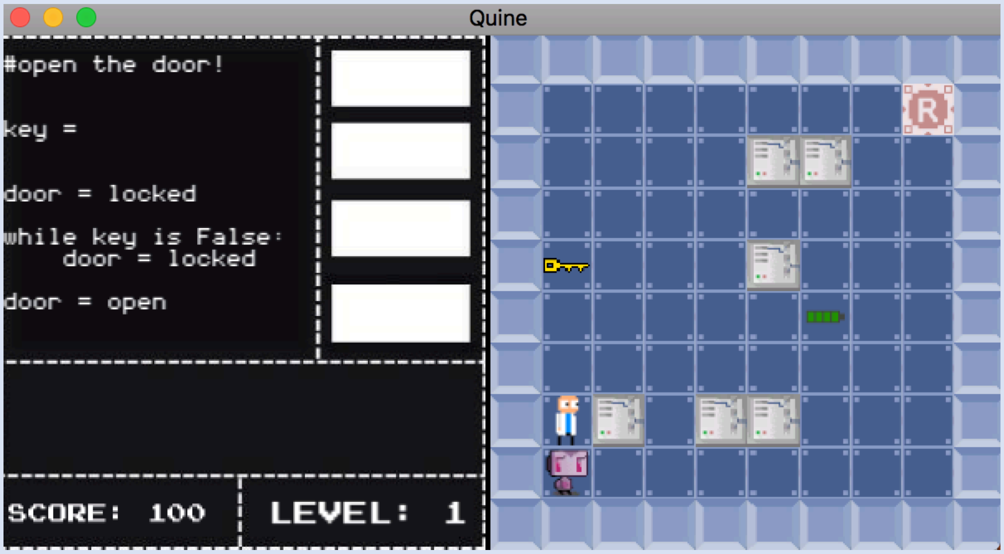
Many proposed changes were discussed in the redesign. The most important change was the change in learning goals and the structures of the levels. After spending time revamping the design for the levels, I was able to come up with a more concise and organized method for teaching players about programming. Focusing on the very basics such as variable assignments and Boolean logic in the beginning levels was essential in order to not overwhelm any players. It was important to actively keep in mind the target audience of my game: students without any computer science or programming background.

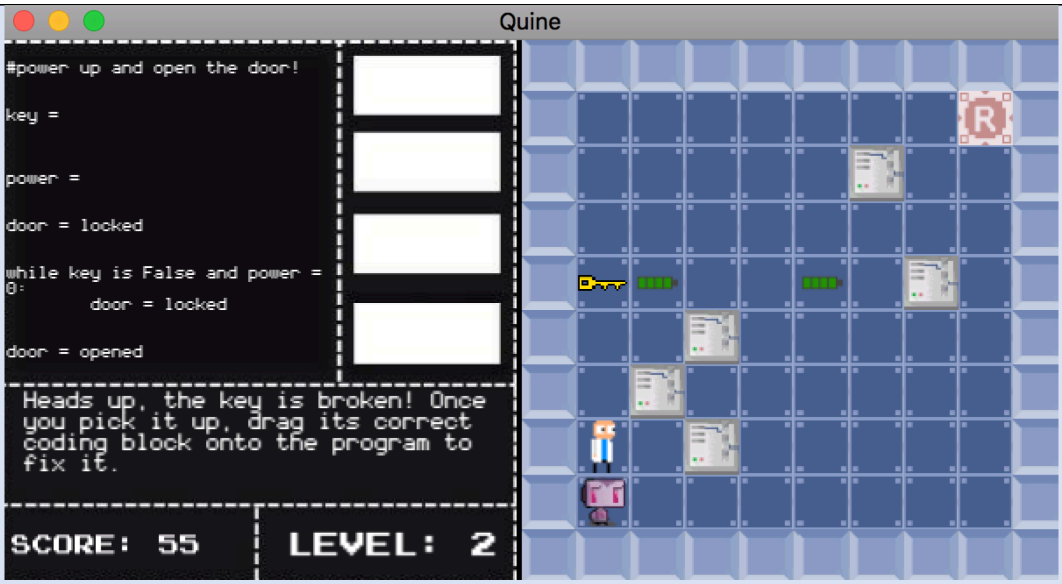
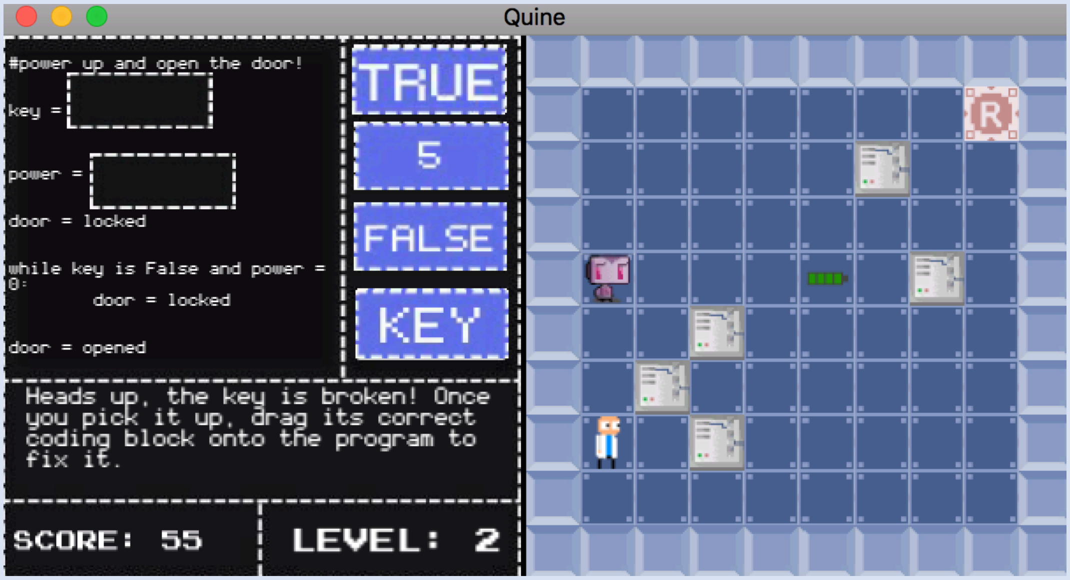
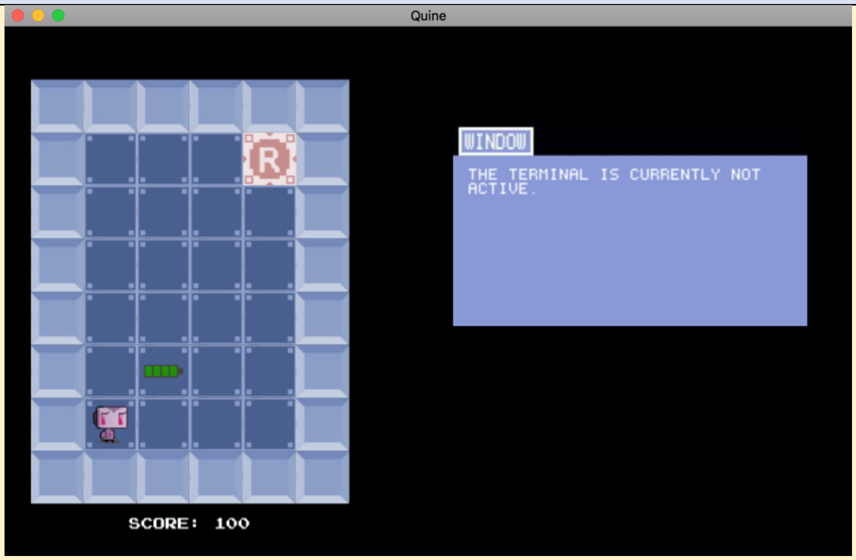
Furthermore, adding instructions, menus, and interactable items gave the game a more “game-like” aesthetic. The ability to interact with items and effectively “hack” them allowed me to bridge the gap between the educational aspect of Quine and the gameplay. This was all done in hopes of being able to motivate players to continue playing the game.

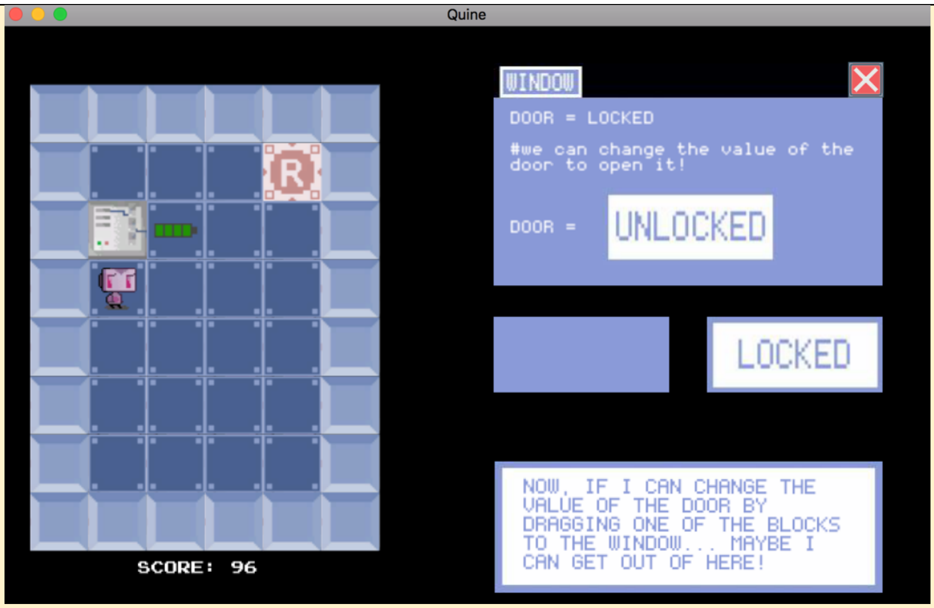
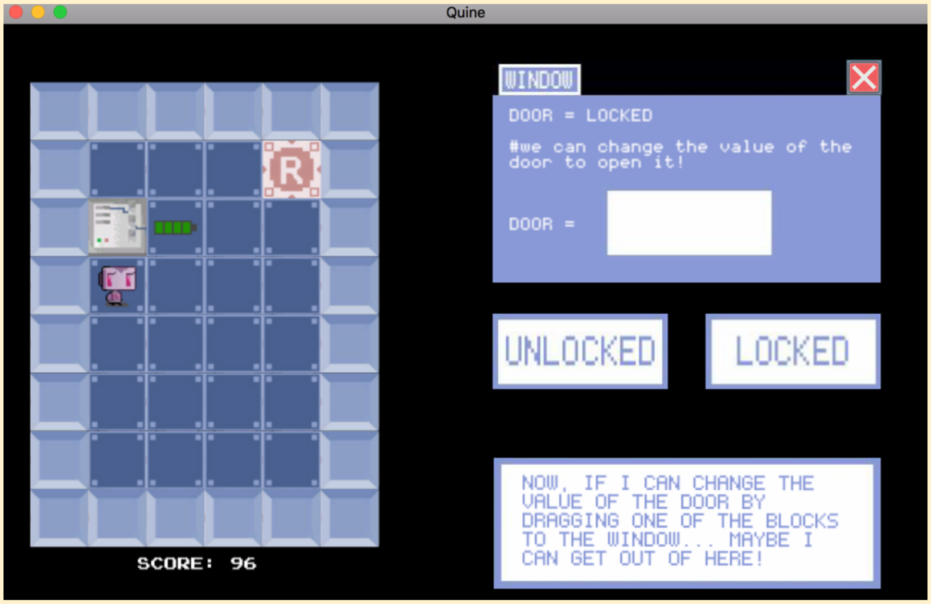
The final playtest was important in seeing how design changes affected my game. Overall, many found the game puzzles to be paced well and found that they had learned something from the game. The increase in the number of users who would play the game as a supplement for learning in the classroom was a good indicator for the effectiveness of the game as a fun learning tool. However, it was clear that many found the game to be boring—in fact, the number of users interested in programming decreased after playing the game. Furthermore, many found the game to still be difficult.

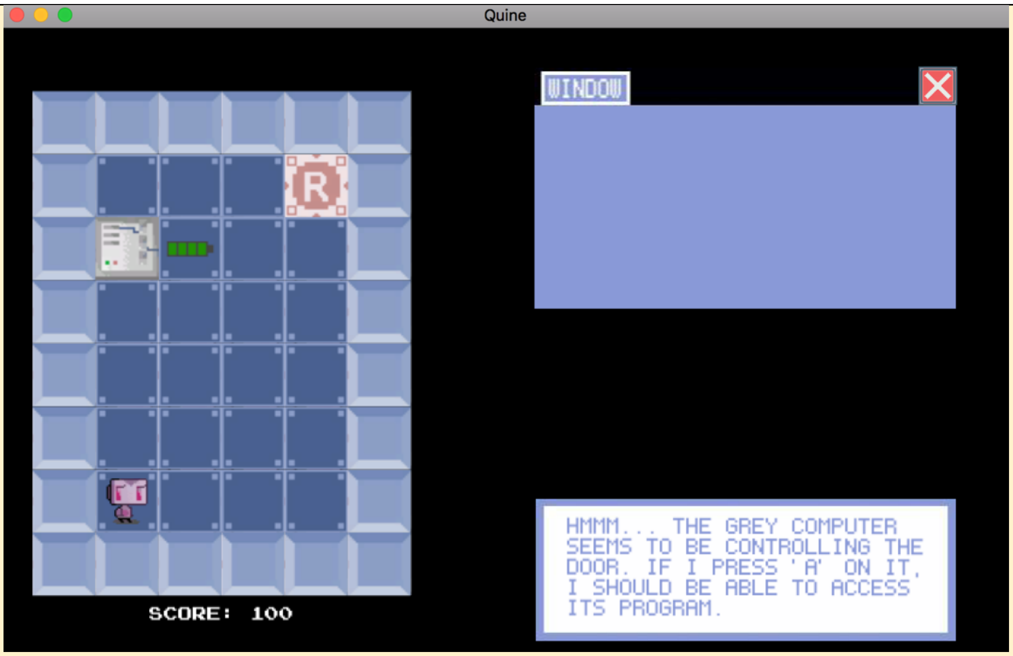
It is clear that more design changes need to be implemented in order to make the game more fun and effective as a learning tool and a possible substitute for classroom learning. More specifically, changes in the visual instructions and level design are essential in order to make the game more fun and accessible. While it is important to note that the main goal of the game is to teach students, being able to make the game more amusing can get more students to feel motivated and interested in the subject. Clear instructions can also have the same impact; if players know what to do, and have a clear idea of what they need to accomplish, they will be motivated to play the game. If more design changes were to be done, a focus on motivation and visual instructions would help.

Appendix A

Level	Design
1	 Quine #open the door! key = door = locked while key is False: door = locked door = open 5 TRUE FALSE ITEM SCORE: 71 LEVEL: 1  Quine #open the door! key = door = locked while key is False: door = locked door = open SCORE: 100 LEVEL: 1

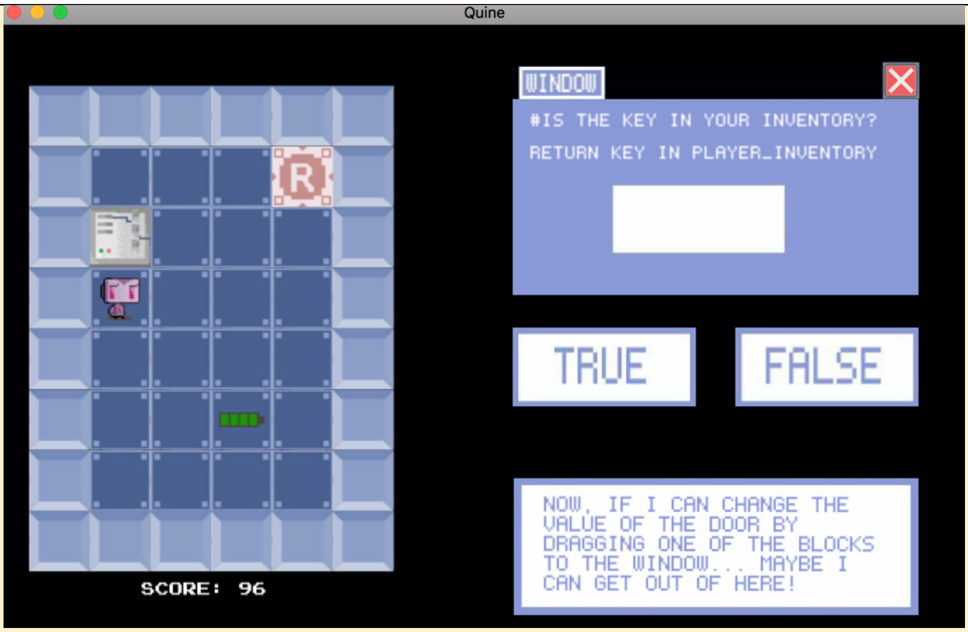
2	 #power up and open the door! key = power = door = locked while key is False and power = 0: door = locked door = opened Heads up, the key is broken! Once you pick it up, drag its correct coding block onto the program to fix it. SCORE: 55 LEVEL: 2  #power up and open the door! key = power = door = locked while key is False and power = 0: door = locked door = opened Heads up, the key is broken! Once you pick it up, drag its correct coding block onto the program to fix it. SCORE: 55 LEVEL: 2
1	 THE TERMINAL IS CURRENTLY NOT ACTIVE. SCORE: 100

2	
	
	DOOR = LOCKED
	#we can change the value of the door to open it!
	DOOR = UNLOCKED
	NOW, IF I CAN CHANGE THE VALUE OF THE DOOR BY DRAGGING ONE OF THE BLOCKS TO THE WINDOW... MAYBE I CAN GET OUT OF HERE!
	SCORE: 96
	
	
	DOOR = LOCKED
	#we can change the value of the door to open it!
	DOOR =
	NOW, IF I CAN CHANGE THE VALUE OF THE DOOR BY DRAGGING ONE OF THE BLOCKS TO THE WINDOW... MAYBE I CAN GET OUT OF HERE!
	SCORE: 96



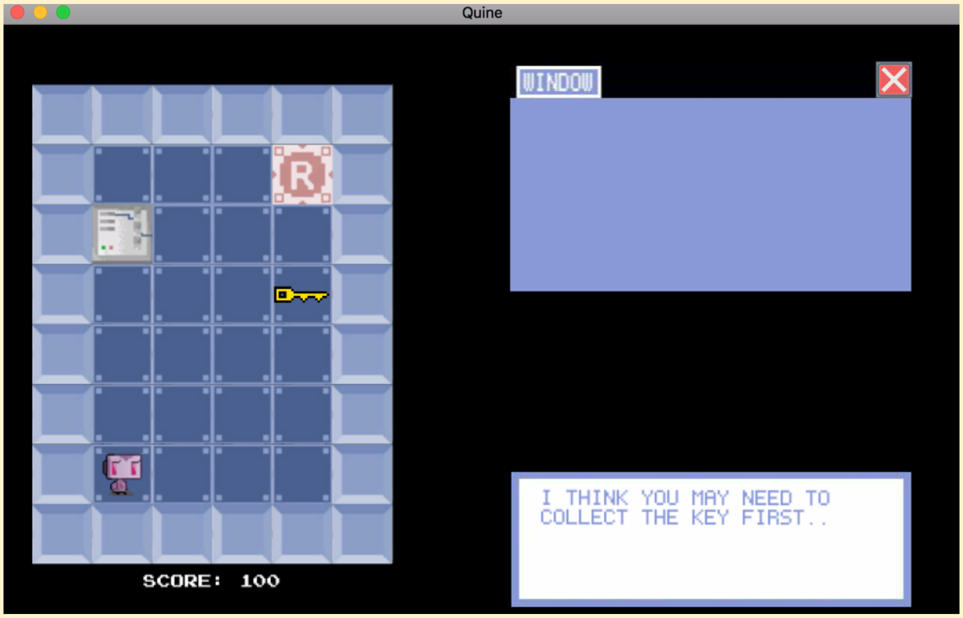
HMMM... THE GREY COMPUTER SEEMS TO BE CONTROLLING THE DOOR. IF I PRESS 'A' ON IT, I SHOULD BE ABLE TO ACCESS ITS PROGRAM.

3

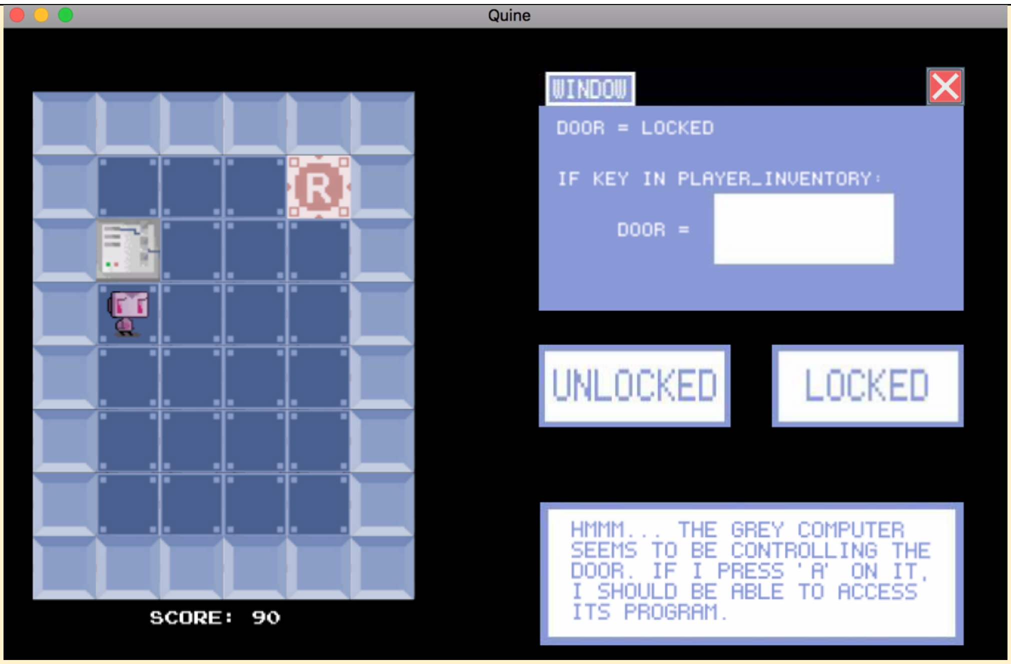


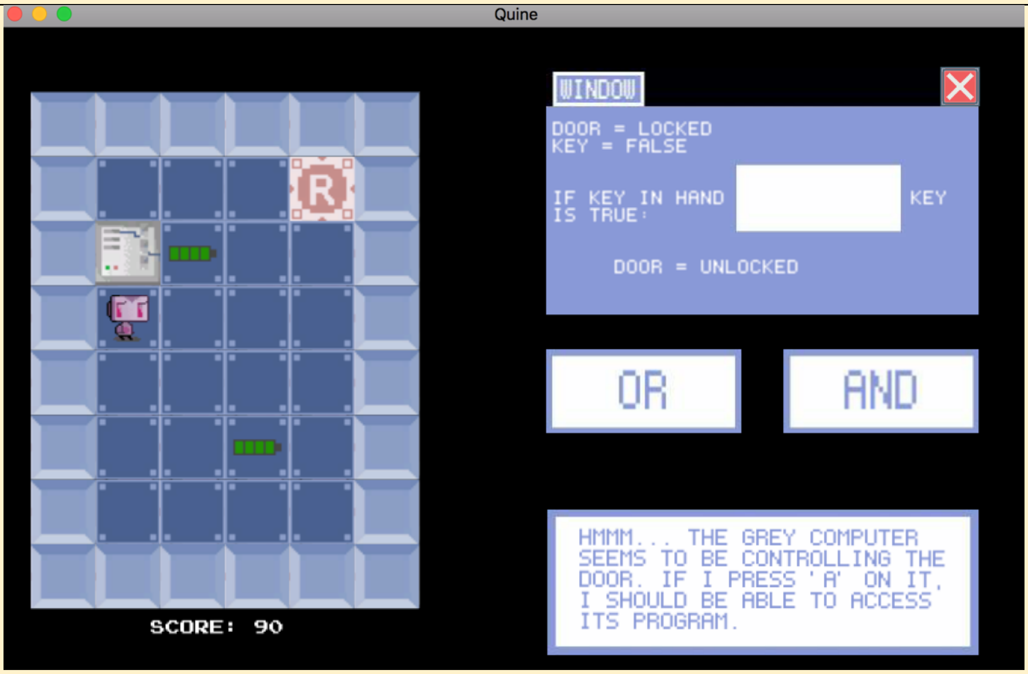
```
#IS THE KEY IN YOUR INVENTORY?
RETURN KEY IN PLAYER_INVENTORY
```

NOW, IF I CAN CHANGE THE VALUE OF THE DOOR BY DRAGGING ONE OF THE BLOCKS TO THE WINDOW... MAYBE I CAN GET OUT OF HERE!

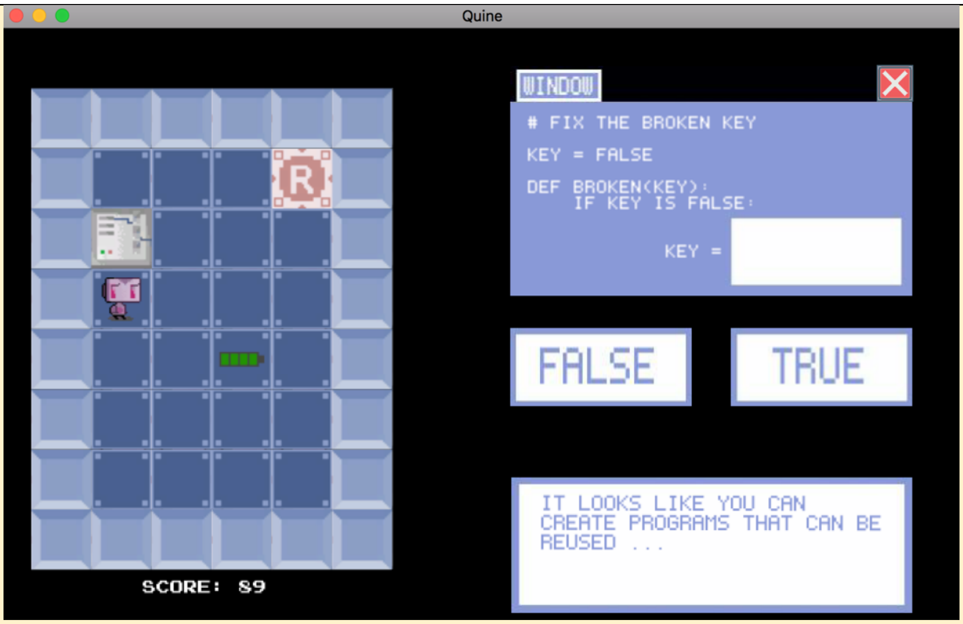


I THINK YOU MAY NEED TO COLLECT THE KEY FIRST..

4	 DOOR = LOCKED IF KEY IN PLAYER_INVENTORY: DOOR = HMMM... THE GREY COMPUTER SEEMS TO BE CONTROLLING THE DOOR. IF I PRESS 'A' ON IT, I SHOULD BE ABLE TO ACCESS ITS PROGRAM.

5	
	
	DOOR = LOCKED
	KEY = FALSE
	IF KEY IN HAND IS TRUE: KEY
	DOOR = UNLOCKED
	HMMM... THE GREY COMPUTER SEEMS TO BE CONTROLLING THE DOOR. IF I PRESS 'A' ON IT, I SHOULD BE ABLE TO ACCESS ITS PROGRAM.

6

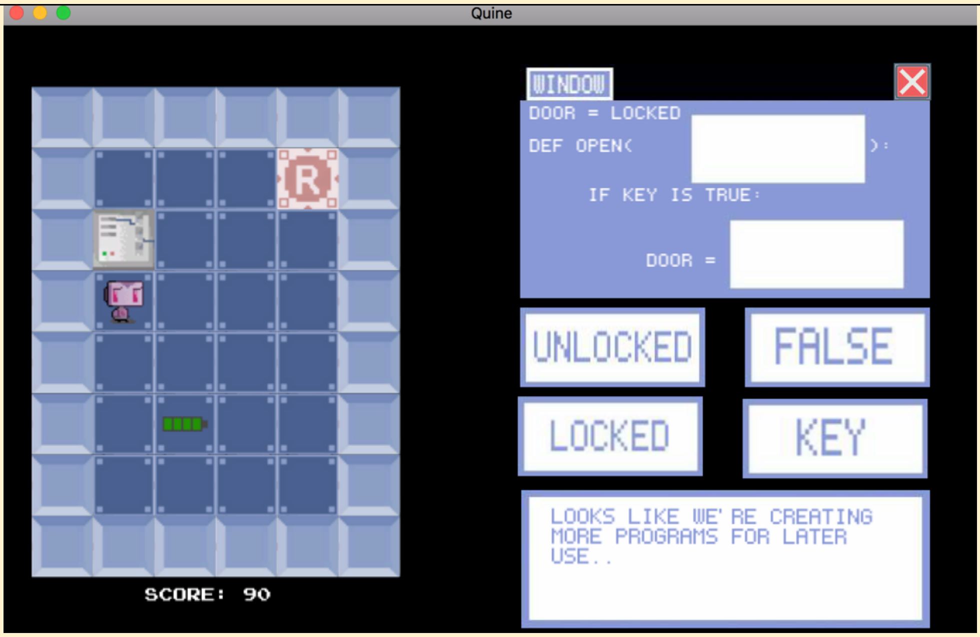


```
# FIX THE BROKEN KEY
KEY = FALSE
DEF BROKEN(KEY):
    IF KEY IS FALSE:
          KEY =
```

IT LOOKS LIKE YOU CAN CREATE PROGRAMS THAT CAN BE REUSED . . .

SCORE: 89

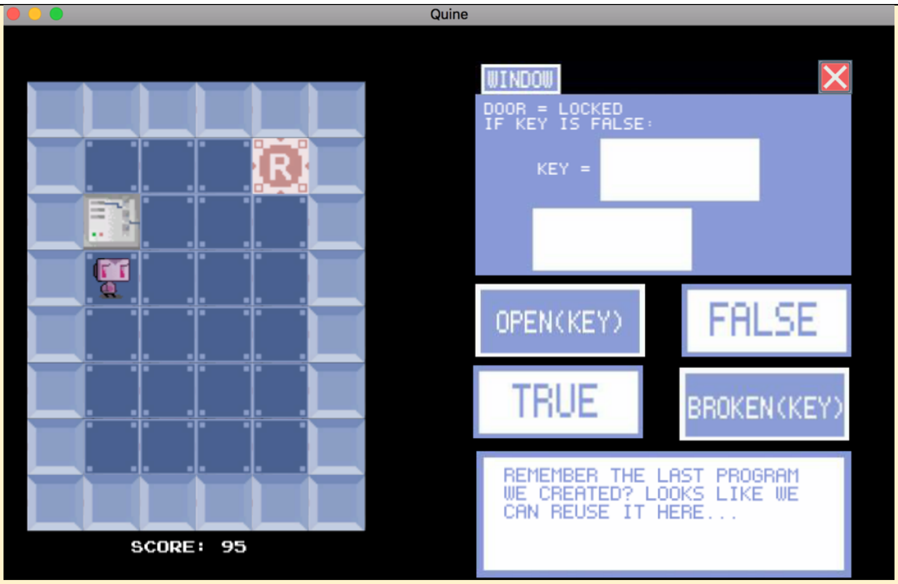
7



```
DOOR = LOCKED
DEF OPEN(          ):
     IF KEY IS TRUE:
          DOOR =
```

LOOKS LIKE WE'RE CREATING MORE PROGRAMS FOR LATER USE..

SCORE: 90

8	 DOOR = LOCKED IF KEY IS FALSE: KEY = REMEMBER THE LAST PROGRAM WE CREATED? LOOKS LIKE WE CAN REUSE IT HERE... SCORE: 95 Quine WINDOW OPEN(KEY) FALSE TRUE BROKEN(KEY)	