CS 111 Final Project  
Self Assessment

# Group

*Who’s in our group?*

1. Charles Zhou ([charleszhou2023@u.northwestern.edu](mailto:charleszhou2023@u.northwestern.edu))
2. Sean Carlson ([seancarlson2024@u.northwestern.edu](mailto:seancarlson2024@u.northwestern.edu))

# Goals

The primary goal of our game is to make the adventure fun and user friendly for our users.

The secondary goal would be to demonstrate our understanding and mastery of the Racket language.

Obviously, getting a good grade is important as well, so we made sure all requirements are met.

# Lessons learned

*What went right?*

Effective collaboration is the most crucial factor in every group project. Looking back, we have always been ready on time, finished our task within the schedule, and communicated efficiently. Because our team is small, we learned that being reliable and proactive is vital in group projects. Be a fountain, not a drain.

We used git for collaboration, and it helps us to work on separate tasks simultaneously using the branches. This facilitates our development process, and we are able to go back to the last version when the new version does not compile.

We also created a google docs outline our plan. We created a map of the rooms and a tree of all the types, which helped tremendously during the development process by providing us a structured, holistic overview that helped us to understand of our code, especially in remembering all the fields and methods.

We deepened our understanding of the racket language during the process, especially the racket structure types and their relations. All the valuable experiences we gained from completing this program will undoubtedly help us in our future CS endeavors.

*What went wrong?*

We had some issues when copy and paste some code from google docs to the racket IDE because of the punctuations.

*What do you wish you knew when you started?*

We wish we knew that racket has printf and sleep commands when we first started. They were very helpful during the development process.

# Annoying grading bookkeeping

## Types

*What are the types you added, and what are they for?*

1. Keycard: for door access control.
2. Securitycam: for room surveillance (triggers losing scenario when on and the player is not disguised).
3. Food: enables eat to regen the hunger value of the player. A surprise item can be hidden inside.
4. Beverage: enables drink to regen the thirst value of the player.
5. Storage: storage container that can hold item. It has open and close commands.
6. Disguise: a type of thing the player can wear to disguise from security cameras.
7. Holygrail: a type of thing the play takes to win the game.
8. Laptop: has a battery level and allows hacking.
9. Laptopcharger: allows laptop charging.
10. Poweroutlet: allows laptop charging when a charger and a laptop is present.

## Fields

*What are the fields you added, what types did you add them to, and what are they for?*

1. Owner: keycard-owner The owner of the keycard.
2. Access-level: keycard-access-level determines the access level of the keycard
3. Privilege: keycard-privilege additional privilege for partial access level keycards.
4. Status: securitycam-status Boolean for on-off status of the camera.
5. Satiety: food-satiety Satiety value for food.
6. Satiety: beverage-satiety Satiety value for beverage.
7. Batterylevel: laptop-batterylevel Battery level of the laptop.
8. Hunger: person-hunger Hunger value of the person.
9. Thirst: person-thirst Thirst value of the person.
10. Hp: person-hp Health value of the person.
11. Disguised?: person-disguised? The disguise status of the person.
12. Lockstatus: door-lockstatus The lock status of the door.
13. Takable?: thing-takable? Determines if a thing is takeable by the player.
14. Trap: room-trap Determines if the room is a trap that triggers losing scenario.

## Procedures

*What are the procedures you added or significantly modified from their original form, and what are they for?*

1. new-keycard Creates a new keycard.
2. new-securitycam Creates a new securitycam.
3. new-food Creats a new food. Will check input satiety value to make sure it’s in range.
4. new-beverage Creats a new beverage. Will check input satiety value to make sure it’s in range.
5. new-storage Creates a new storage.
6. new-disguise Creates a new disguise.
7. new-holygrail Creates a new holygrail.
8. new-laptop Creates a new laptop.
9. new-laptopcharger Creates a new laptopcharger.
10. new-opweroutlet Creates a new poweroutlet.
11. mystatus Displays player’s hunger, thirst, and hp
12. check-hunger Displays player’s hunger
13. check-thirst Displays player’s thirst
14. check-health Displays player’s hp
15. check-battery Displays the laptop’s battery level
16. close Closes the storage.
17. hack Hacks the camera and changes its status when laptop is present and charged.
18. Remove-disguise Get rid of the disguise and updates disguised? status.
19. gracefuldeath Makes the death graceful. Moves the player to heaven and display a line to see if player wants to try again.
20. endcredit Displays an awesome end credit.
21. loadingscreen Displays a good-looking loading screen. Because why not?
22. bargraph Displays a cool bargraph for hunger, thirst, and battery values.
23. health-regen Regens the health when hunger and thirst levels are both full.
24. update-stats Updates the player’s hunger and thirst when an action is performed. Check if player is disguised and add extra penalty. Able to trigger losing scenario when hp reaches 0.
25. nearby? Check if something satisfying predicate is in reach (in the room and inventory)
26. have-a-in-room? Check if something satisfying predicate is in the room (does not check inventory)

## Methods

*What are the methods you added or significantly modified from their original form, what types were they added to, and what are they for? Note that if you have three different methods for the same generic procedure, list each one separately.*

1. Thing (take thing) Modified to exclude player inventory, implemented logic for takable? field, and included hunger and thirst value update.
2. Door (go door) Added keycard access control, check if a room is a trap behind the door (triggers losing scenario), check if there’s a working camera in the room behind the door and if player is disguised(triggers losing scenario), and included hunger and thirst value update.
3. Keycard (examine keycard) Return the proper keycard description based on owner names.
4. Securitycam (examine securitycam) Return the proper securitycam description based on cam working status.
5. Food (eat food) Updates the hunger and health value. Also, a surprise element is implemented if there’s an item inside a food object.
6. Beverage (drink beverage) Updates the thirst and health value.
7. Storage (open storage) Opens a storage container.
8. Disguise (wear disguise) Changes the disguised? value of the player. Included hunger and thirst value update. Take the disguise item if it is not in player inventory.
9. Holygrail(take holygrail) Triggers winning scenario.
10. Laptop (charge laptop) Updates the laptop batterylevel when a charger and an outlet is present. Included hunger and thirst value update.
11. Laptop (examine latop) Returns the proper description based on laptop batterylevel.
12. Object (eat food) Returns a line for nonfood objects.
13. Object (drink beverage) Returns a line for nonbeverage objects.
14. Object (open storage) Returns a line for nonstorage objects.
15. Object (wear disguise) Returns a line for nondisguise objects.
16. Object (charge laptop) Returns a line for nonlaptop objects.

## Total stuff we built

*66*