Our Game

In our game, you control a robot trying to escape a factory by collecting keycards to open the factory gate and escape. The robot needs to avoid the factory guards that will try to capture them, as well as EMPs that will drain their energy, potentially disabling them if they don't have enough.

Tutorial video: https://www.youtube.com/watch?v=SG6JRByGeVI
Presentation slides: https://docs.google.com/presentation/d/1BZv8tDSP1jWj_HH5wjLPl4cc30eEd4fRJYyWEiXlKUE/edit?usp=sharing

Changes From Initial Design

We followed most of the functional specifications that we initially defined, with notable exceptions being the exact implementation of the pause menu, display of the in-game controls, and short cutscenes.

We initially specified that the pause menu should be invoked via clicking a button and also display controls while paused, but switched to having it be invoked by pressing 'p' on the keyboard while playing the game. The rest of our interface was controlled by keyboard, so we wanted to maintain consistency by having the pause menu also controlled via keyboard. We didn't display any controls on the pause menu because we decided this would look too cluttered and potentially confuse the player.

We omitted cutscenes when starting and winning the game due to time constraints, deciding that this feature was not particularly important for overall functionality. Additionally, it could have been annoying to players that wanted to simply access the gameplay as quickly as possible.

In terms of the internal design specified by our initial UML class diagram, we made several adjustments after noticing missing functionality from classes, inefficiencies in communication between classes, and to account for the specific javax and swing libraries we used for visualization. Of particular note, we added a collision property to tiles, which allowed us to create walls by specifying a different tile number, rather than implementing walls as objects on the map tiles. We also abstracted certain duties to separate classes such as CollisionChecker, AssetSetter, and KeyHandler (for keyboard inputs), increasing cohesion.

Lessons Learned

- **Gain understanding of implementation details before creating UML diagrams.** Our initial UML diagram could have been more accurate to our final implementation if we had spent more time learning about the libraries we used for graphics and thinking more about how specific program states and actions would be handled by classes prior to making it.
- **Frequent communication and acting quickly are key in software development.** Feedback between group members was given often, which meant that after "completing" a feature, additional development time was required to discuss and act on feedback. This necessitated acting quickly and communicating frequently to ensure we could implement feedback and keep the project on-schedule.
- **Ensure every developer understands the full code.** The nature of asynchronous development means that developers often work with only parts of the project at a given time. However, this shouldn't mean that a single developer's understanding should be limited to only certain parts of a project. When transitioning to work on different features, and during debugging and testing, understanding of other parts of the code may become crucial to completing a task correctly and in the most efficient way. We realized later in the project, when adding our final features and debugging, that our individual understandings of the existing code was sometimes limited, resulting in inefficient approaches to adding new features and writing tests. If we had left more descriptive comments and communicated more of the technical details of our earlier features, we would not have had this problem.

Build Artifacts

Prebuilt artifacts for our game can be found in the prebuilt-artifacts folder of our project. Details for how to generate and run the artifacts can be found in our project's README.md file.