Team 14
Kevin Flynn
Andrew Callahan
Hayden Reich
Jonathan Vu

Title: Side Scrolling Game

1. **Features implemented:**

ID	Requirement
UR-002	High scores should be saved with associated initials
UR-003	The user should be able to manuever the character with arrow keys
UR-005	The user should be able to pause the game
UR-007	The user should be able to pick up power ups
UR-008	The user should be able to attack enemies
UR-009	The user should be able to start the game from the menu
FR-002	The player should be able to move around level
FR-003	The enemy's health should decrease when in attacked by player
FR-004	The enemy should disappear when defeated
FR-005	The enemy will be able to attack the player
FR-006	The enemy will be able to move

FR-007	The player should die and game ends when health is depleted
NR-001	The game window should be moveable
NR-002	The system should interact with mysql when transferring account data
NR-003	The game should run smoothly at every game state

2. Features Not Implemented

UR-001	The game state should be able to be saved under current account
UR-004	The user should be able to change the difficulty of the game
UR-006	The admin should have access to each account created, as well as being able to manage all high scores
UR-010	Create Account
FR-001	The difficulty should have variable levels

3. Class Diagrams

See New_Class_Diagram.pdf for the final version of our class diagram, and Old_Class_Diagram.pdf for the version of our class diagram turned in as part of Part 2.

Our final class diagram was significantly different from our earlier submission. Aside from eliminating the Account classes entirely, we have a much more extensive diagram after redesigning and implementing. Some things remained similar, the Game class was central and remained central to diagram as a driver class for the others. Screen expanded dramatically to reflect the complexity of it as an object. It in fact expanded too much and is ripe for refactoring as it grew larger than it should have. The interaction between menu and game remained, but the GameObject class was eliminated in favor of an abstract sprite class and its derivatives: Character, Enemy and Powerup, and the generic Environment object and its derivative Destructable. Enemy and Powerup both get their own derivative types as well, further expanding our final class diagram.

4. Design Patterns

We used a simple form of the Chain of Responsibility pattern to handle key press actions in the game. This allowed us to pass key press events from the active listener class, Screen, to the first handler class, a private class internal to Screen called TAdapter. This would then process events further and then continue the chain by passing the event to the second handler class, in this case a private Character instance. See COR_Diagram.pdf (only relevant members and methods shown). There were a couple other places in the program where we could use design patterns after refactoring.

While we didn't implement it, we did make use of the List classes iterator function, which is a library implementation of the iterator pattern. That doesn't really count though.

On refactoring, we could most likely change the Screen class to make use of the state pattern, as the behavior of the class changes based on a few different factors. In the current implementation, both a paused and running screen could be states, but in future versions we would then have the ability to easily add additional states as needed. See Screen_State.pdf.

5. Reflection

Having walked through the process of designing and implementing a program, there are a few points that stand out. The first is that design is much easier when there are clear goals and a well-defined direction. In our project, we started out with the idea of writing a side scrolling video game, but that idea evolved as we discussed individual abilities, time constraints, and user experience. We wound up creating more of a platformer than a side scrolling game, with the enemy/player interaction being very different than what we originally envisioned. This meant that some of our original requirements were either no longer relevant, or the wording no longer exactly represented our project goals. Our class diagrams and use case diagrams also diverged due to this evolution.

If we had started with a more concrete idea, it may have streamlined the process of translating design into implementation, as the redesigns would have been much smaller in scope. Despite this, the general blueprint of the classes and functionality offered by the early design process, still made the implementation vastly easier and more efficient than would have been the case had we omitted the design process entirely.