BATTLE
SET
HEINZE COURVILLE

Overview

Based on the card game *Set*, BattleSet was developed to allow players to play head-to-head against the computer. The idea is to match up 3 cards before the computer does. If you are successful, the computer takes damage; if not, you take damage. When too much damage is taken, that player dies and the game ends.

The software was developed from the ground up, starting with designing the game, utilizing UML and flow charts for sequence diagraming. The use of flow charts was our biggest asset. It allowed the team to visualize the flow of the game, define classes and relationships between classes, was a major troubleshooting tool by identifying missing steps, and helped clarify steps where a team member may have used a "leap of faith" technique that methods or classes would properly communicate.

After creating the UML and flow charts, a Gant chart was created to keep the team on track and possibly identify imbalances in our plan where we may have given too much or too little time to accomplish tasks.

Once the design was created and set to paper with a 90 percent solution -- we, with our lack of experience, decided that a 90% solution was our threshold to start coding -- we started to code. We used a technique similar to the construction of a building, starting with a foundation, the design phase; then the framing, our coding of the very basics of the game; and then to the finishing steps of proper flow and responses that make the game fun to play.

We started with the anti-design pattern or the god pattern. We lumped all the basic coding into one class. Although this eventually took up more time than if we used separate classes, this allowed us to gain insight to how well we planned and designed the game. With a basic game running successfully, we spread out the code into several different classes and began to expand upon the basic game, by adding players, battle logic, ability to use more than one deck, and a startup screen, just to name a few.

As mentioned earlier, our debugging was improved with the use of flow charts. Because software engineering can cover some concepts instead of something tangible, flow charts allowed the team to visualize the weakness and ensure all members are on the same page at the same time. This allowed us to brainstorm corrective actions and "see" if they worked before we spent quality time coding in a hope-and-see technique.

Contents

A Unique Twist on a Simple Card Game

Battle Element

Logic that allows a person to go head-to-head with a computer opponent. This allows competition and pride to increase the fun of playing.

A.I. Player

After a random amount of time, if the human player has not successfully designated a set, the computer takes over, identifies a true set, decrements the human player's health, and then allows the human to attempt a chance to ID a set. If no set is available, the computer will automatically add cards to the table in groups of three. Then, it allows the human a chance.

Health

Both the human and the computer start out with full health. Then based on the cards chosen, or in some cases not chosen, the amount of health is modified. When a player's health reaches zero, that player dies, and the game is over. The modifications are as follows: improper "set," their health is ecremented by one.When a player chooses the "No Set" option and there is no set available, the opponent's health is decremented by one.

When a player chooses the "No Set" option and there is a set available, that player's health is decremented by two.

When a player chooses a proper "set," the other player's health is decremented by one.When a player chooses

Multi Deck Games

Play with one deck of 81 unique cards or play with as many decks as you wish.

Name

Personalize the game by adding your name into the game. If you don't want to see your name in the game, a default name will be used.

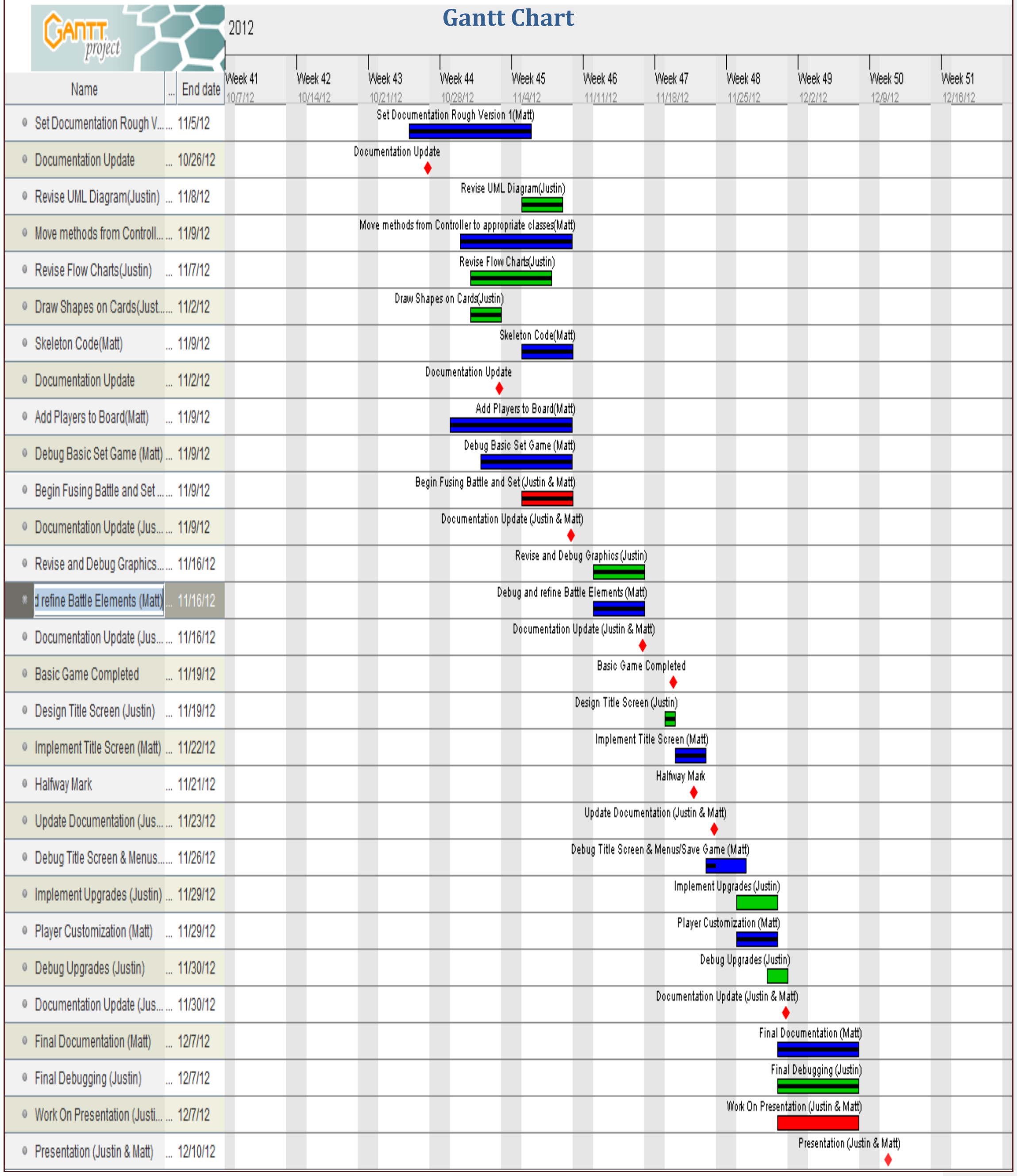
Level

The game has several levels. As each level increases, the random amount of time you have until the computer takes over decreases. Also, the human player has the ability to choose the game play level they wish to start out at.

Title Screen

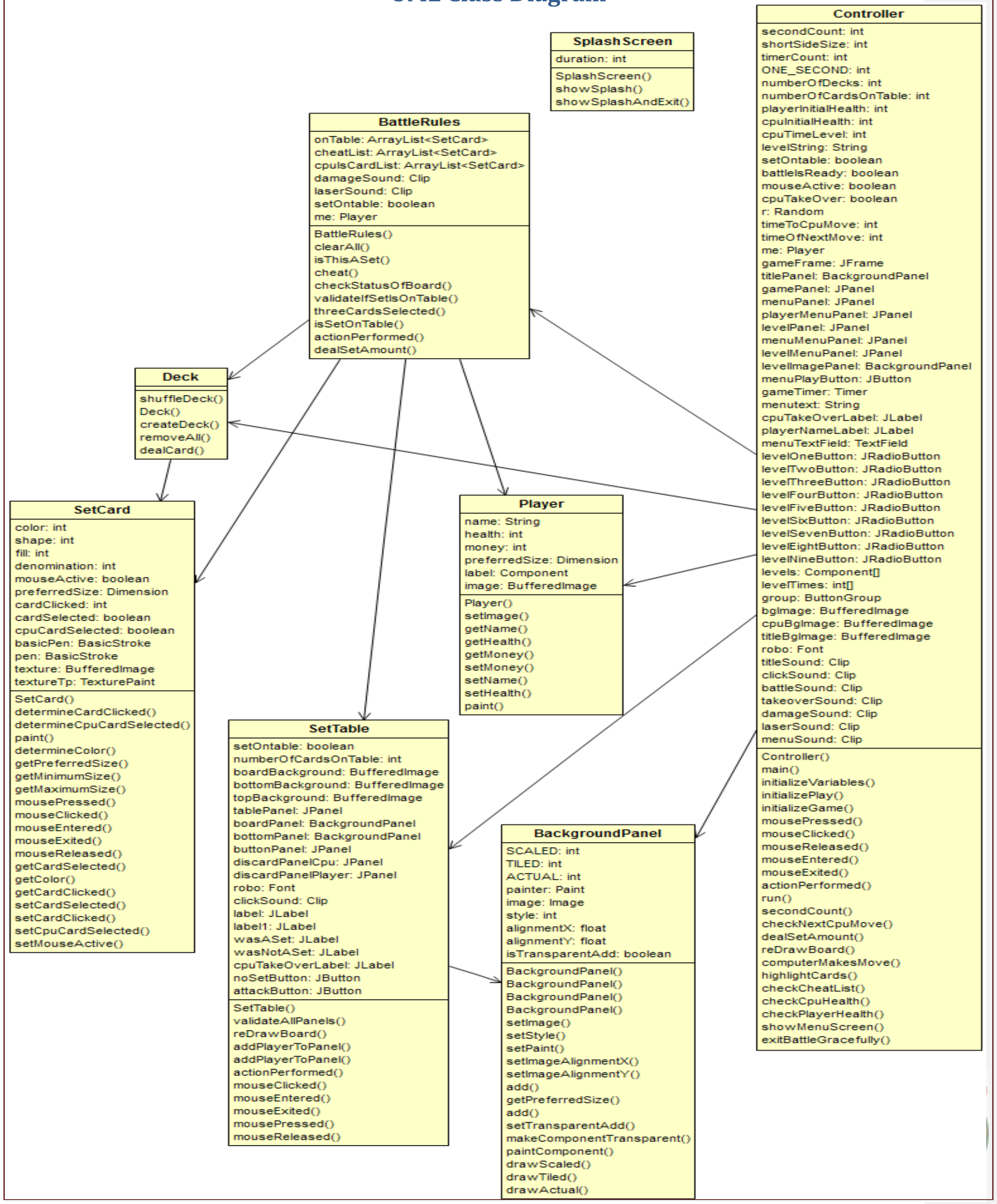
Adding a touch of class, this is where you get to choose the level of game play and add your name.

Gantt Chart



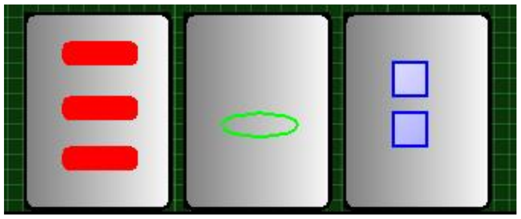
Name	...	End date
Set Documentation Rough V...	...	11/5/12
Documentation Update	...	10/26/12
Revise UML Diagram(Justin)	...	11/8/12
Move methods from Controll...	...	11/9/12
Revise Flow Charts(Justin)	...	11/7/12
Draw Shapes on Cards(Just...	...	11/2/12
Skeleton Code(Matt)	...	11/9/12
Documentation Update	...	11/2/12
Add Players to Board(Matt)	...	11/9/12
Debug Basic Set Game (Matt)	...	11/9/12
Begin Fusing Battle and Set ...	...	11/9/12
Documentation Update (Jus...	...	11/9/12
Revise and Debug Graphics...	...	11/16/12
d refine Battle Elements (Matt)	...	11/16/12
Documentation Update (Jus...	...	11/16/12
Basic Game Completed	...	11/19/12
Design Title Screen (Justin)	...	11/19/12
Implement Title Screen (Matt)	...	11/22/12
Halfway Mark	...	11/21/12
Update Documentation (Jus...	...	11/23/12
Debug Title Screen & Menus...	...	11/26/12
Implement Upgrades (Justin)	...	11/29/12
Player Customization (Matt)	...	11/29/12
Debug Upgrades (Justin)	...	11/30/12
Documentation Update (Jus...	...	11/30/12
Final Documentation (Matt)	...	12/7/12
Final Debugging (Justin)	...	12/7/12
Work On Presentation (Justi...	...	12/7/12
Presentation (Justin & Matt)	...	12/10/12

UML Class Diagram

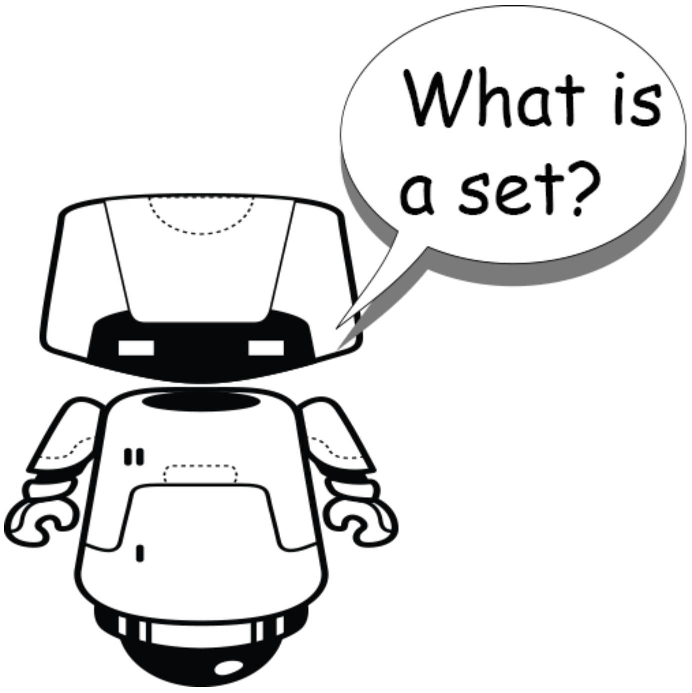


Users Guide

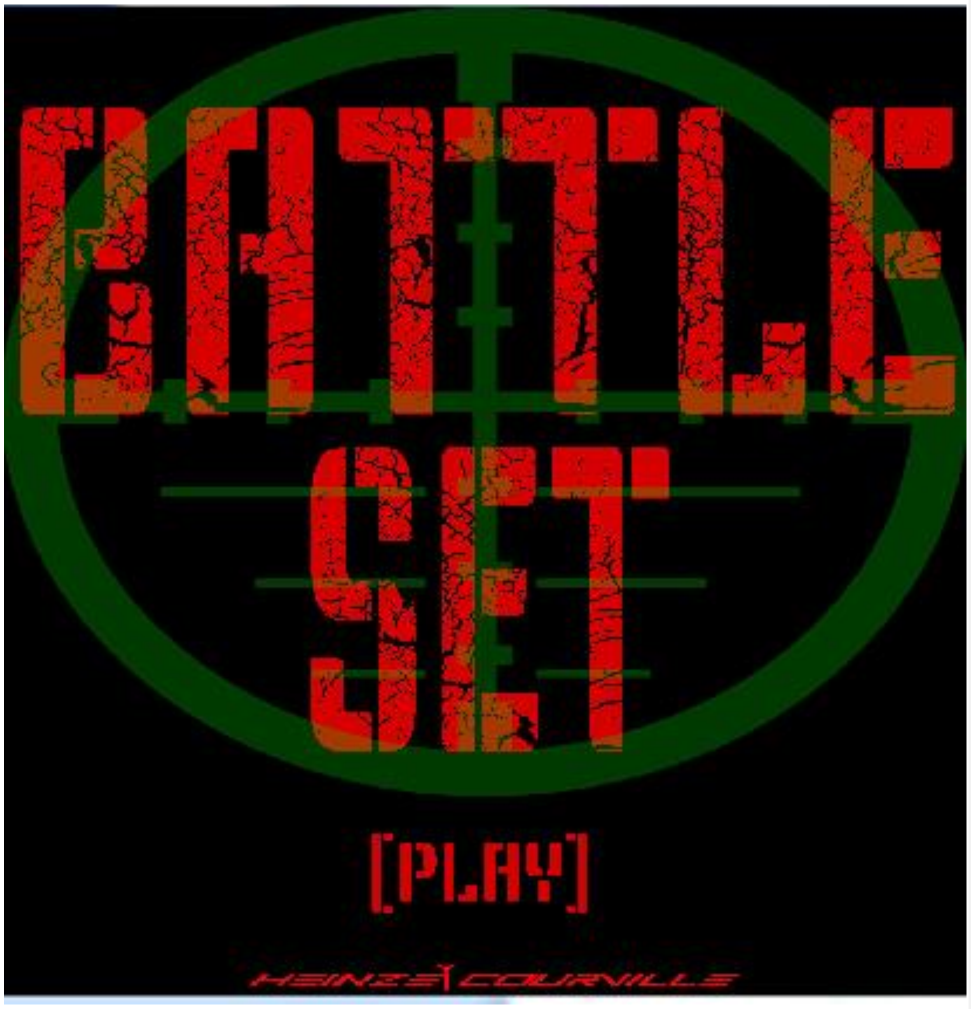
Identifying a Set



There are 81 cards to a deck. Each one is unique and has four elements: shape, color, amount, and fill. Your job is to identify a "set." A "set" consists of three cards, and each of the four elements must all be the same or all must be different. A proper set may have no elements the same, all the elements the same, or any combination between the two, as long as the individual elements are either the same or completely different.

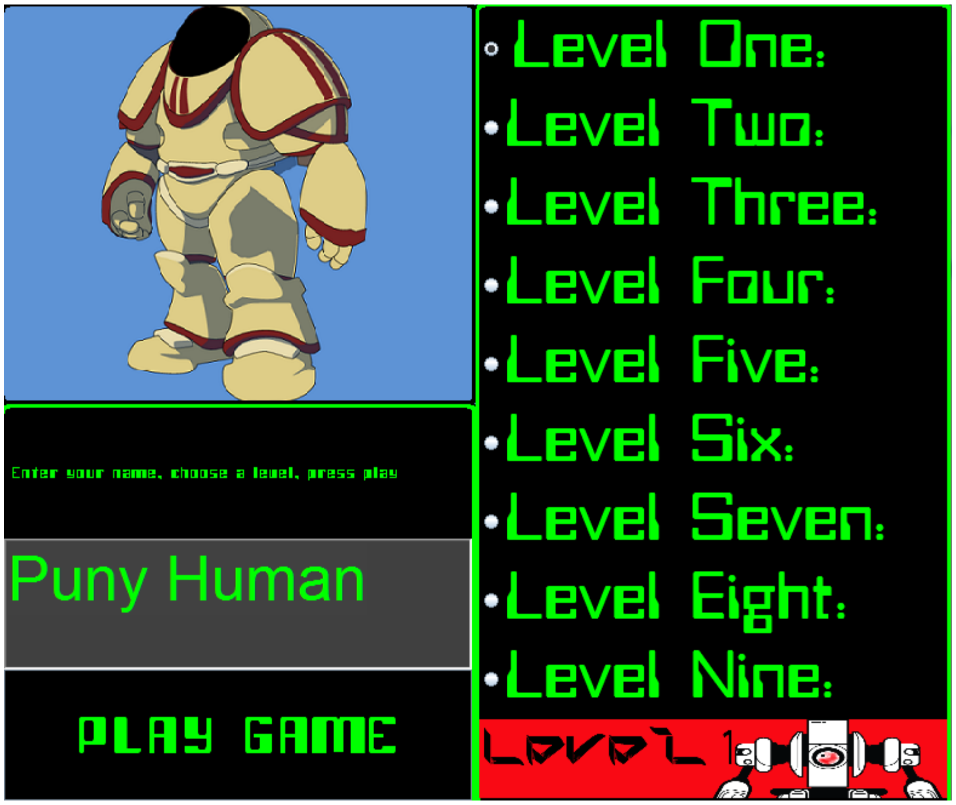


Title Screen



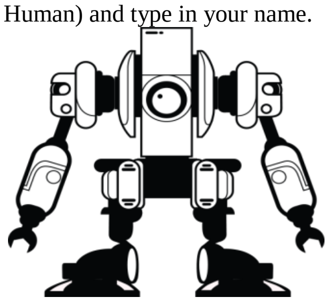
To play, simply click anywhere on the play screen with the mouse, or sit back and enjoy the music as your prepare to do battle!

Menu Screen



Name

If you wish to add your name in the name box, delete the default name (Puny Human) and type in your name.



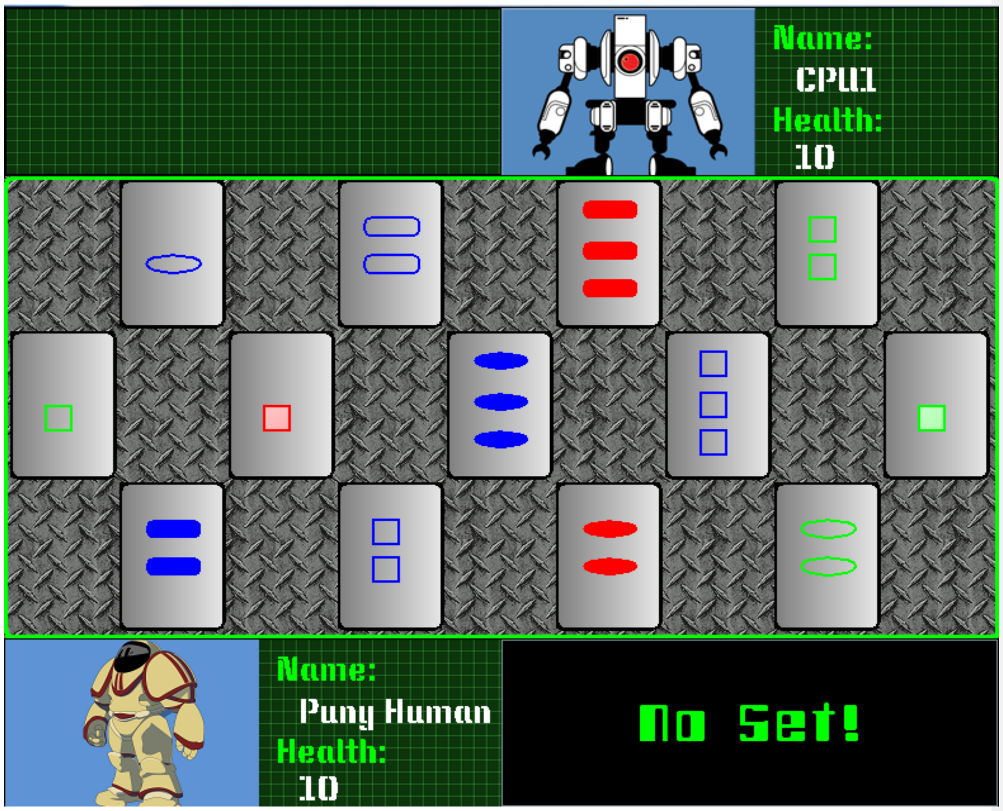
Level Select

Choose a level by clicking on the button next to the level you wish to play at. Each level button will change the picture in the bottom right hand corner indicating your next opponent.

Play Button

Click the play button to start the battle !

Battle Screen



Once the game starts, you have two options.

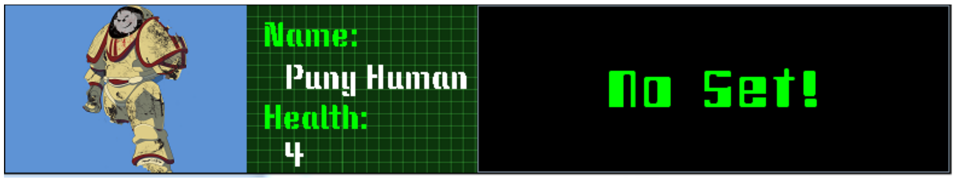
No Set Button if you think that there is no "set" available on the table, you can choose the "No Set" button. If you are correct, your opponent takes damage. If not, you take twice the damage.

Set identify the "set" by clicking on three cards. If your cards are a "set," your opponent takes damage. If they are not a set, then you take the damage.

Remember, time is ticking. The longer you take to identify a proper set, the greater the chances are that your opponent will take over.

Warning: you opponent is never wrong.

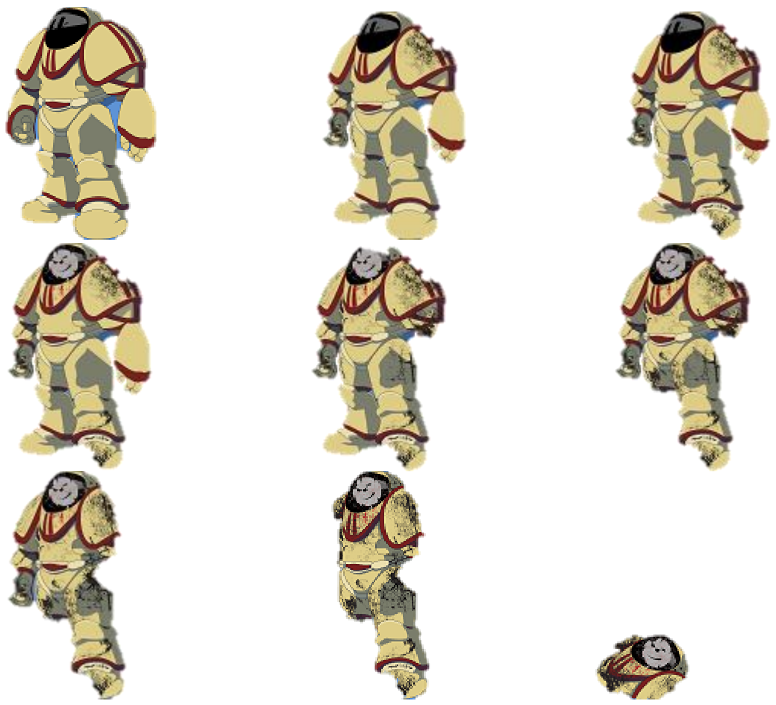
Player Health



Health

Your health is indicated in the lower left hand corner, if it reaches zero you have died and will be prompted to start again.

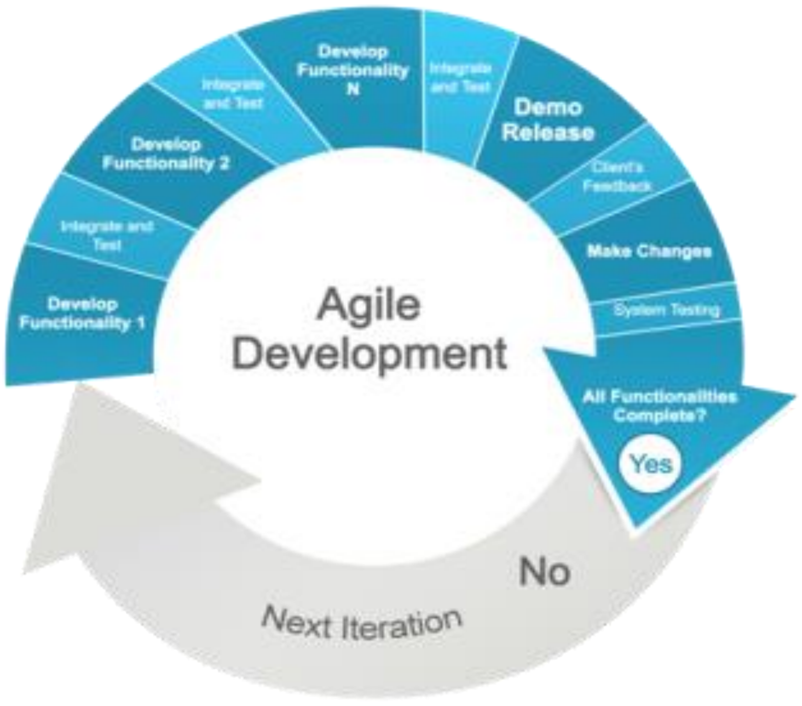
As you take damage your picture will show the increasing damage that you are taking



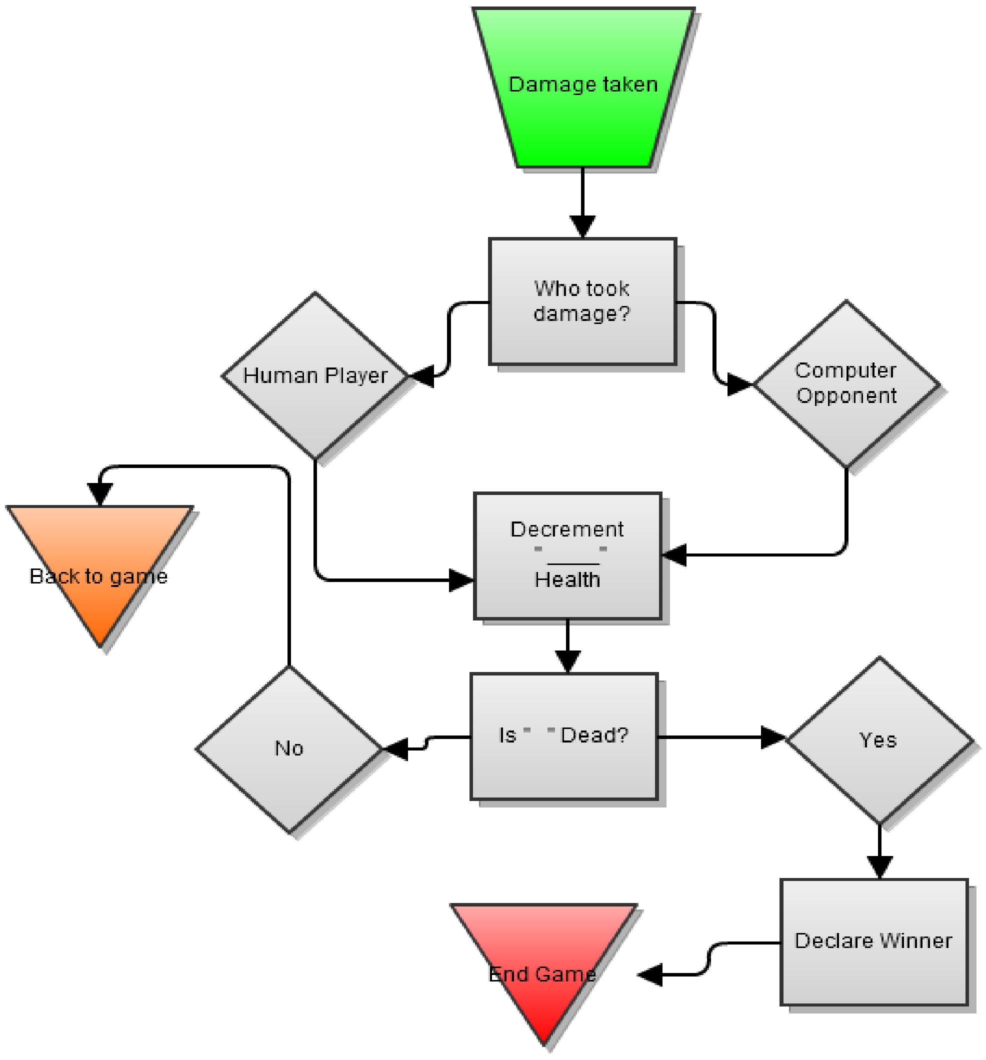
Development Process

Done in Phases

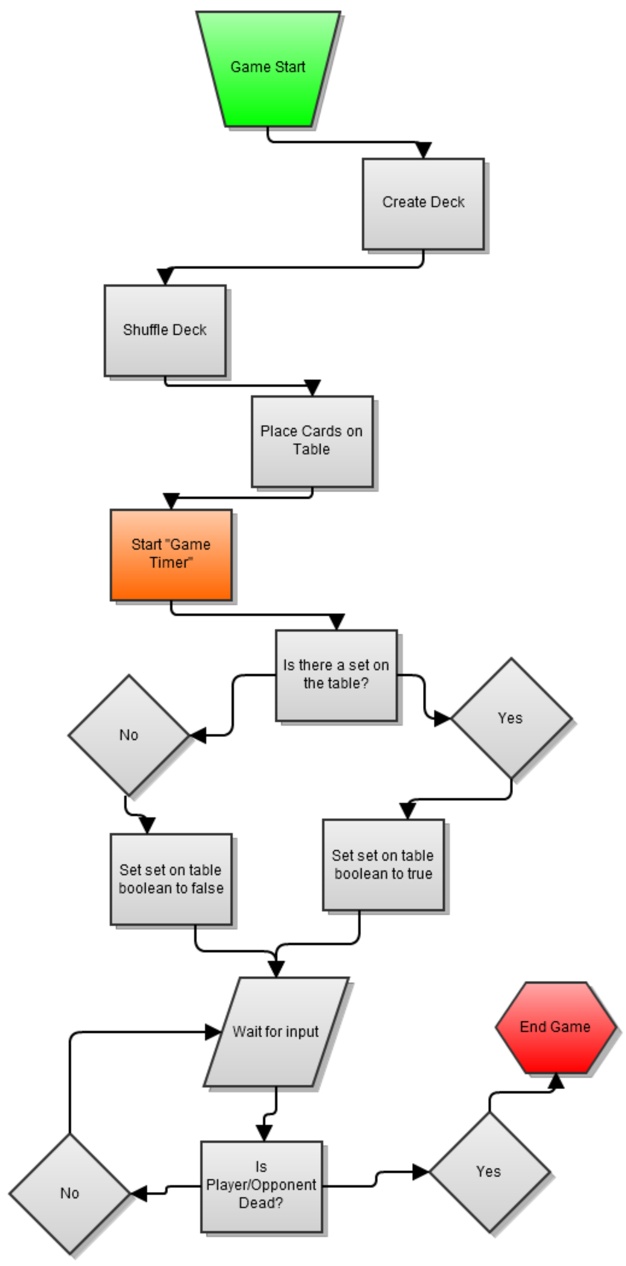
1. **Flow Charts**
2. **Pseudocode**
3. **Work out kinks in pseudo code**
4. **Modularized implementation/ testing**



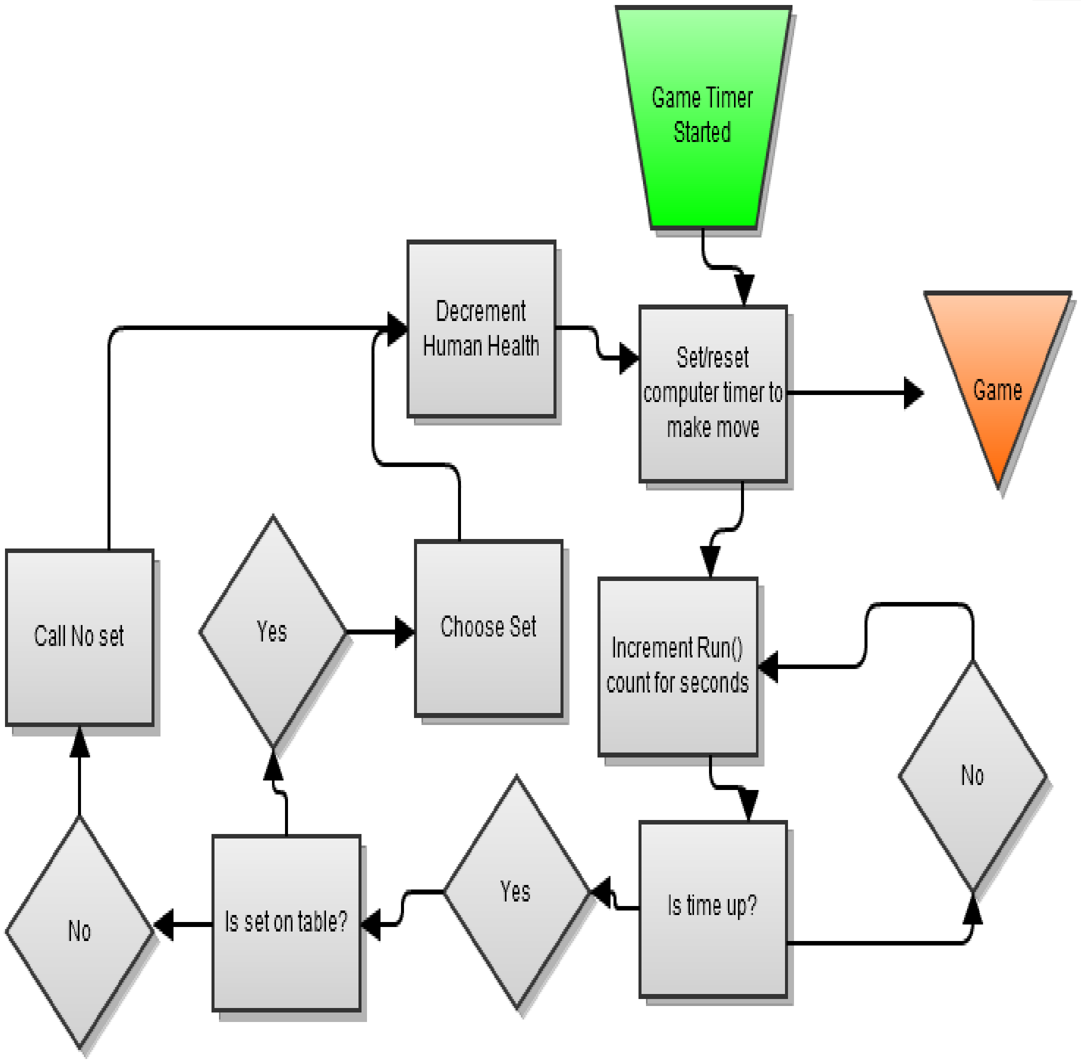
Battle Damage



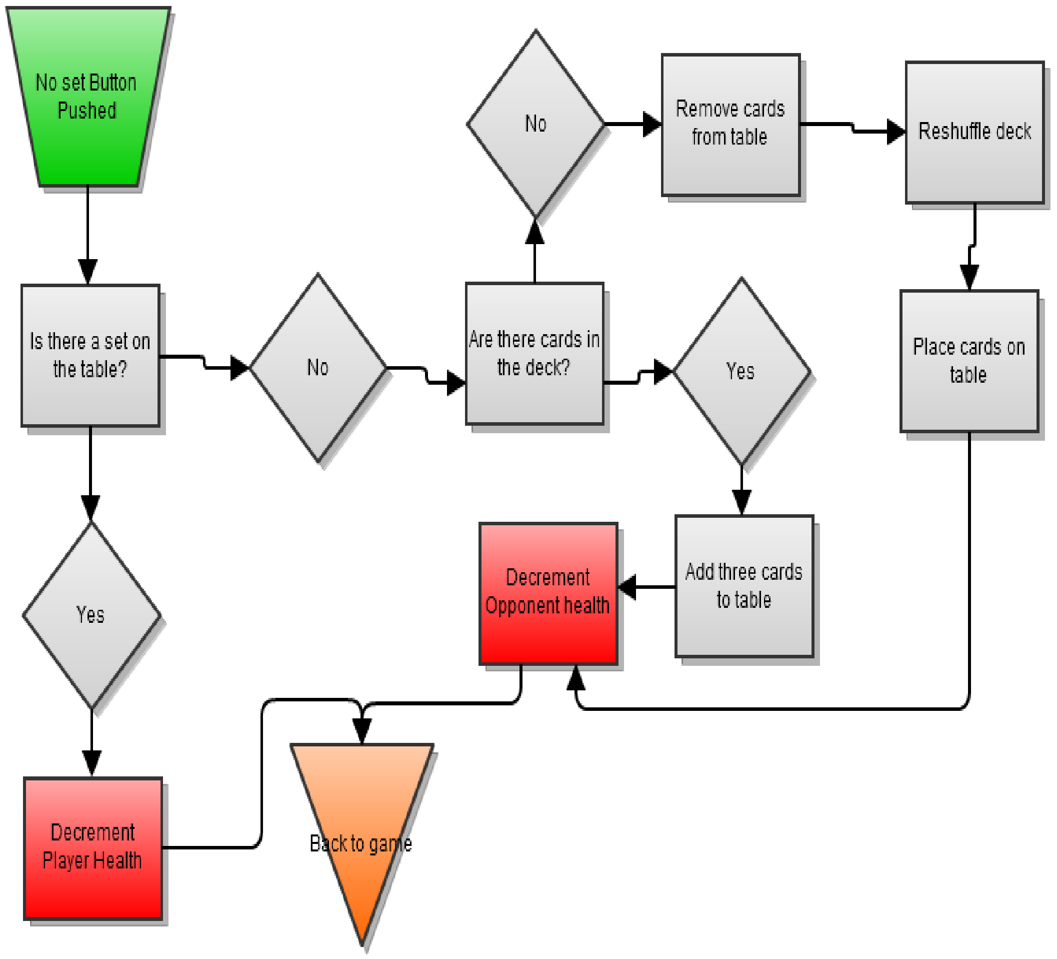
Basic Game


Game Start
Create Deck
Shuffle Deck
Place Cards on Table
Start "Game Timer"
Is there a set on the table?
No
Yes
Set set on table boolean to false
Set set on table boolean to true
Wait for input
End Game
Is Player/Opponent Dead?
No
Yes

Game Timer


Game Timer Started
Decrement Human Health
Set/reset computer timer to make move
Game
Call No set
Yes
Choose Set
Increment Run() count for seconds
No
No
Is set on table?
Yes
Is time up?

“No Set” Pressed



Versions and Testing

Testing

Testing was done in three basic phases.

Phase 1

was continuous. As the code was developed, other code was written to allow that portion to run and to verify the output.

For example, a cheat button was built. The cheat button took the existing code's ability to verify a true "set" was available and then allowed the user to activate the button as if they actually choose three cards in a proper set. This action allowed the verification of the next steps in which the code was to follow after a user had correct input. Next steps included adjusting CPU player's health, checking board to verify a "set" was available based off cards that remained on the table, and replacing cards, if necessary.

During this phase, a Boolean, which told developers if a "set" was available or not; the CPU timer, determining when it took over the game; and the amount of cards in the CPU "set" array was made visible.

Phase 2

was conducted when the basic components to the game were brought together, creating a very basic but functioning BattleSet game. Here, we tested for bugs and attempted to break the game. This phase lasted a couple weeks where the cycle was play game, discover bug, make code corrections/ improvements, play the game. This portion was very much white box testing. To allow possible black box testing, we had a couple testers try the game with only instructions on the objective of the game.

Once we had confidence in a proper functioning game, we progressed into the third phase, Beta testing.

Pahse 3

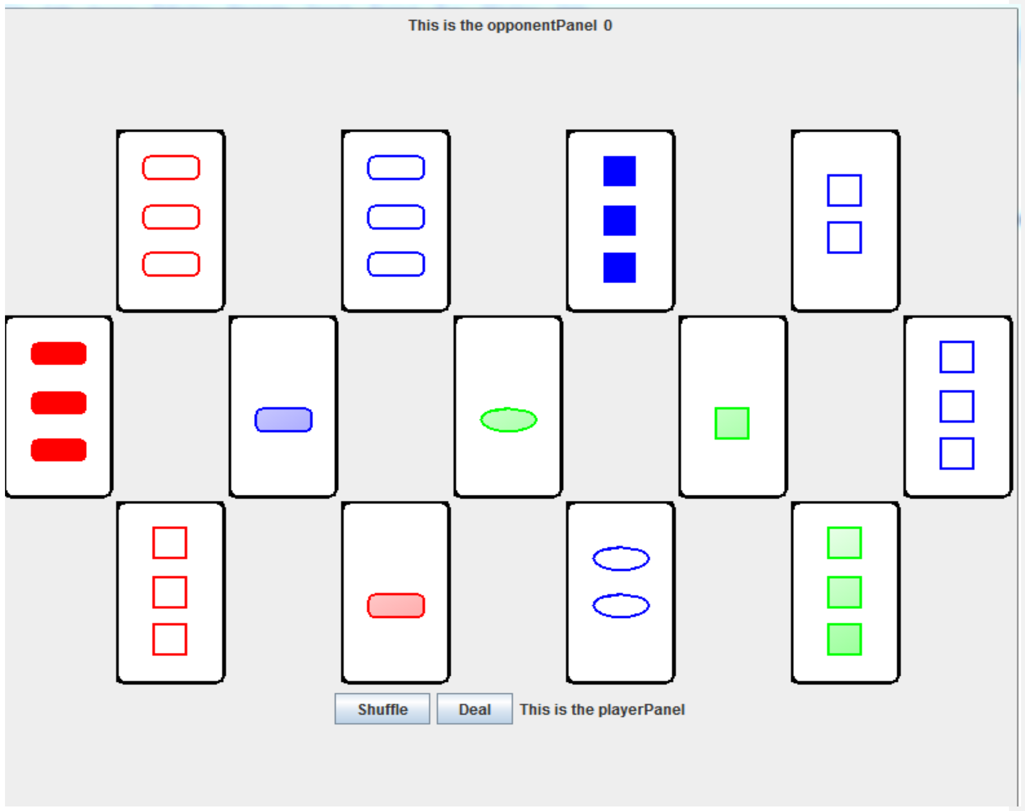
Beta testing was conducted under two basic instructions: one instruction was "this is a game, play it"; and the other instructions were the proper instructions on what was expected and how to play.

The testers ranged in age from 5 years old to 63. The bulk of the testers were teenagers, 14 to 17 years old, and fellow college students.

Beta testing portion was disproportionate black box testing, with some white box as the developers constantly verified the game actions throughout building, testing, and improving the game during its life cycle.

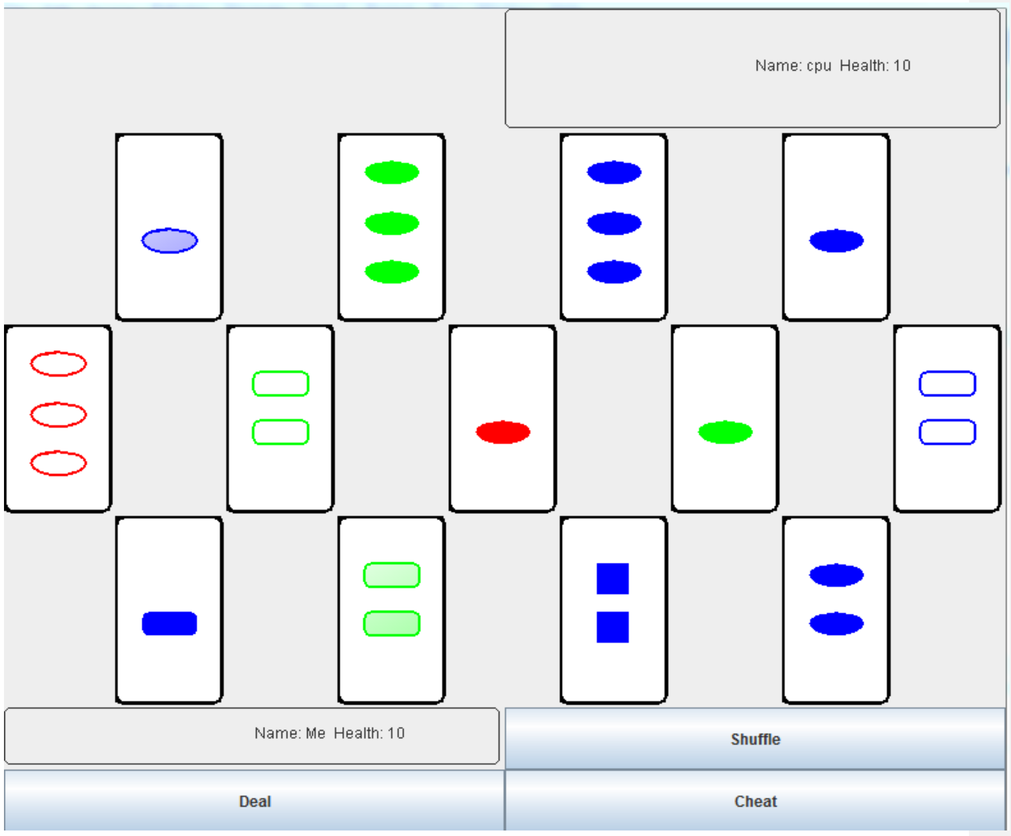
Version 2

Early version simply implement a table with cards on it as well as space for the player and opponent places. The shuffle and deal buttons enabled the game to be debugged and let us know that the shuffle was working correctly and that a card could be dealt and removed from the deck.



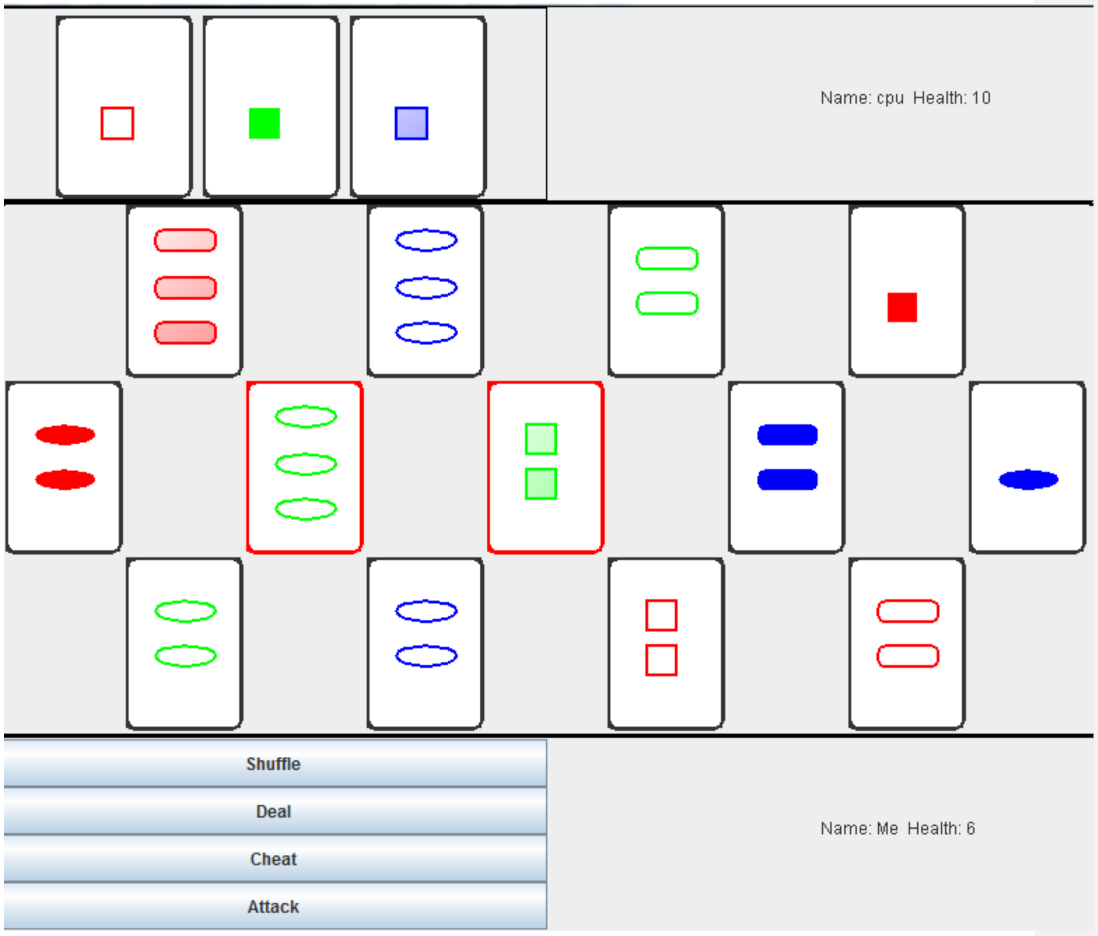
Version 3

Version 3 shows notable changes including the cheat button as well as the player and CPU opponent actually added to the board. The cheat button was a significant part of the game's testing phase as it allowed us to make sure that the CPU was in fact choosing a correct set.



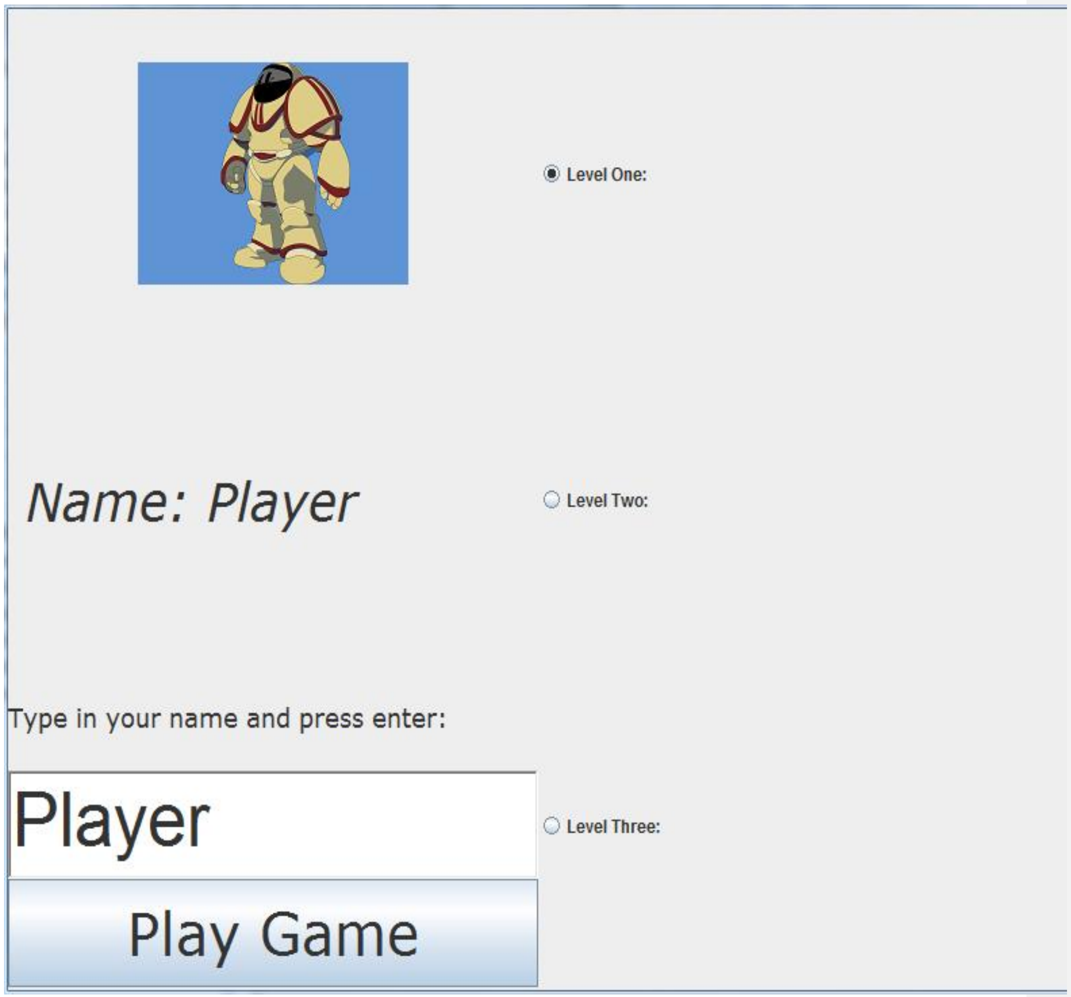
Version 5. 2

Version 5.2 was the next milestone as the cards were now selectable and would be tested as a set. The CPU is now making selections of sets (if there is one on the table of course) and visually showing the selection process.



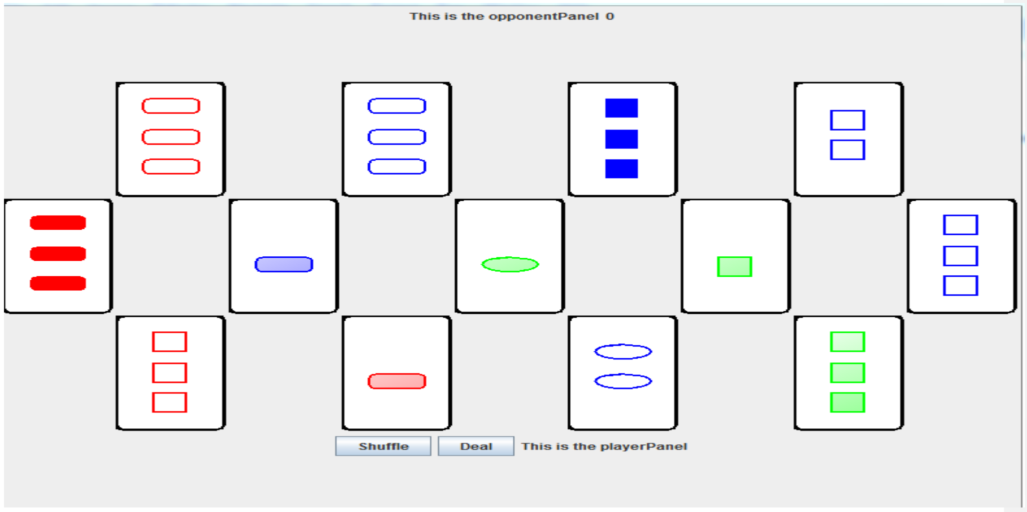
Version 5.5

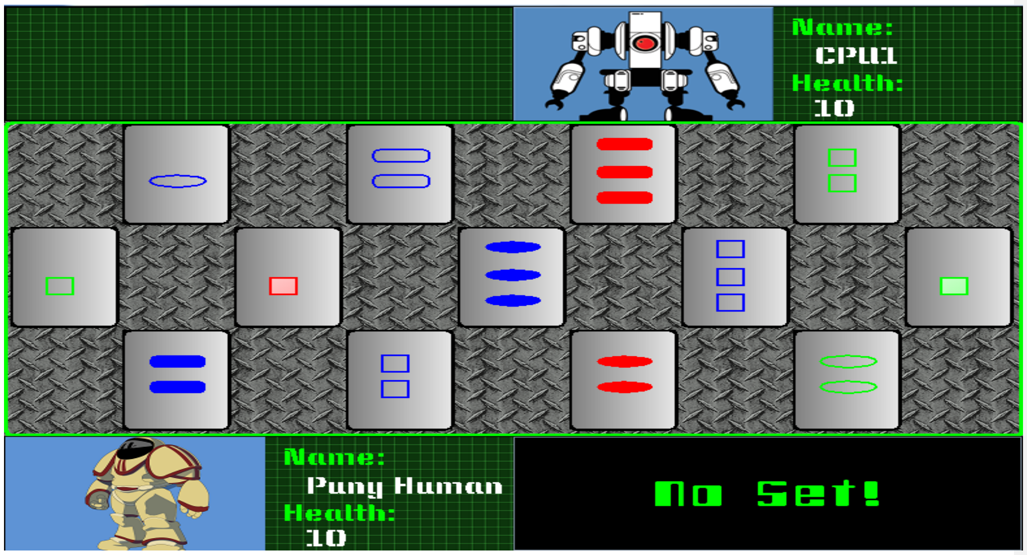
The last notable version was Version 5.5 where a crude title screen was implemented that allowed the player to select a different level and showed a different picture for the player as he took damage.



Version Comparison

Version 2.0 vs. Version 6.1





Interesting Bits of Code

The Cheat Method

The cheat method is actually integral to the game, it is what the CPU relies on to make a move and select a set on the table.

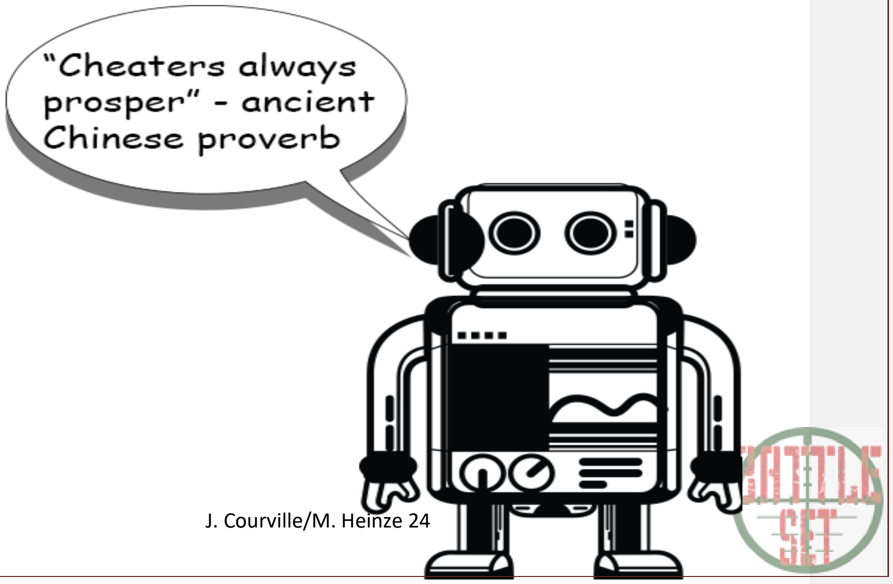
```
    //Cheat method for the computer to be able to choose a set on the table
        public void cheat() {
            cheatList.clear();
            int cardOne, cardTwo, cardThree;
            SetCard card1 = null, card2 = null, card3 = null;
            for (cardOne = 0; cardOne < onTable.size () - 2; cardOne++) {
                for (cardTwo = cardOne + 1; cardTwo < onTable.size () - 1;cardTwo++) {
                    for (cardThree=cardTwo+1;cardThree<onTable.size();cardThree++) {

if(isThisASet((SetCard)onTable.get(cardOne),onTable.get(cardTwo),onTable.get(cardThree
)))
                        {
                            card1 = onTable.get(cardOne);
                            card2 = onTable.get(cardTwo);
                            card3 = onTable.get(cardThree);
                            cheatList.clear();
                            cheatList.add(onTable.get(cardOne));
                            cheatList.add(onTable.get(cardTwo));
                            cheatList.add(onTable.get(cardThree));
                        }//end if
                    }//end cardThree
                }//end cardTwo
            }// end cardOne

        }//end cheat
```

Commented [WU1]: This is important to clear out the old values. A bit of house keeping

Commented [WU2]: Iterate through all possible variations of cards on the table to find a set

Commented [WU3]: Once we have found a set add the three cards that make up a set to the cheatList

Summary: The cheat method is just going to build up the cheatList arrayList



Check Status of Board Method

This method is essentially checking to see if there are cards selected on the table and if so how to handle them. It's main purpose is to keep all of the arraylists current. So every 50 ms the game is doing a little housekeeping and keeping everything nice and clean.

```
public void checkStatusOfBoard(JFrame gameFrame, SetTable table, Player me,
Player cpu){

   //check if 3 cards are selected and if there are handle them and consider
       threeCardsSelected(table, me, cpu);

   //Clear the arraylist of selected cards yes the order of this is important
       isSetList.clear();
       cpuIsCardList.clear();

   // Repaint everything so we don't have anything hanging around
       gameFrame.getContentPane().repaint();

   //Handle selected and unselected cards for the table
       for (int i = 0 ; i < onTable.size() ; i++)
       {
           //If a card is selected, show it
               if(onTable.get(i).cardSelected )
               {
                   if(isSetList.contains(onTable.get(i)) == false)
                   {
                       isSetList.add(onTable.get(i));
                   }

               }//end if
           // If a card is not selected make sure the game and the player
               if(onTable.get(i).cardSelected == false)
               {
                   isSetList.remove(onTable.get(i));
               }

               if(onTable.get(i).cpuCardSelected )
               {
                   if(cpuIsCardList.contains(onTable.get(i)) == false)
                   {
                       cpuIsCardList.add(onTable.get(i));
                   }

               }//end if
           // If a card is not selected make sure the game knows
               if(onTable.get(i).cpuCardSelected == false)
               {
                   cpuIsCardList.remove(onTable.get(i));
               }
       }//end for loop
           // Set the boolean for the game if there is a set on table
       validateIfSetIsOnTable();
}//end checkStatusOfBoard
```

Commented [WU4]: Checking to see if there are any cards selected on the table and if there are who is going to be damaged. We have to pass this in because the method threeCradsSelected() has the logic to decrement health.

Commented [WU5]: So now since this is being called every 50 ms we need to make sure that if a card is selected it isn't getting added to the isSetList hundreds of times. So by simply comparing whats on the table and selected to what is in the selected list we can keep this correct.

Commented [WU6]: Make sure that if a card s not selected it is absolutely not in the isSetList

Commented [WU7]: Checking to see if the CPU has selected anything and if so make sure it isn't being added as a duplicate

Commented [WU8]: Making sure that if a card is not selected by the CPU then it isn't in his selection list

Commented [WU9]: Always a good idea to keep this boolean current

Computer Makes Move Method

The “brain” of the computer making a move. Simply check if there is a set on the table and proceed accordingly.

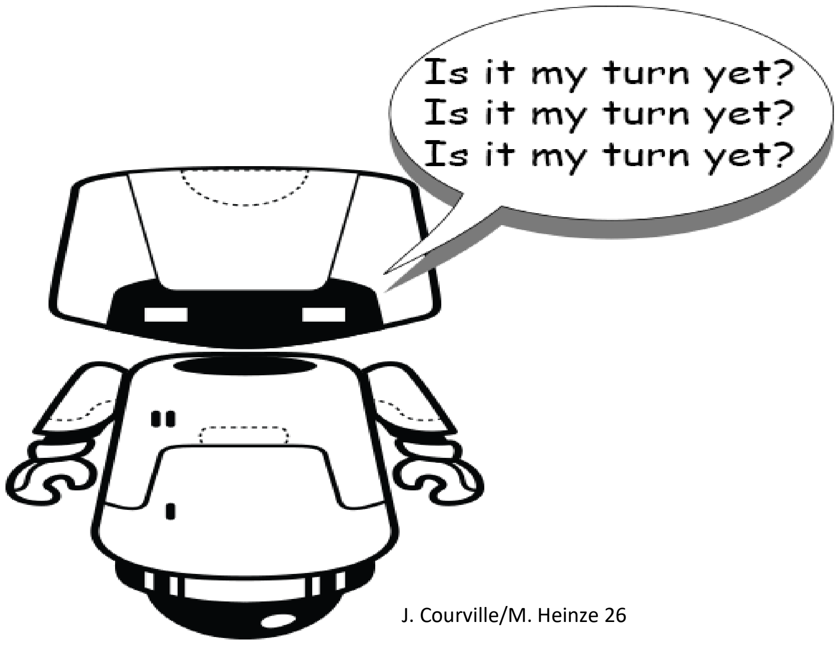
```
    //Computer makes move
        public void computerMakesMove() throws InterruptedException{
            timeToCpuMove = (r.nextInt(10) +1);
            timeOfNextMove = secondCount + timeToCpuMove;
            if(rules.setOntable)
            {
                for(int i =0 ; i < rules.onTable.size();i++)
                {
                    rules.onTable.get(i).setCardClicked(2);
                    rules.onTable.get(i).setCardSelected(false);
                }
            rules.cheat();

            }
            if(rules.setOntable == false)
            {
                me.setHealth(me.getHealth()-2);
                dealSetAmount(3);
            }
        }
```

Commented [WU10]: Variable that is generated by a random number generator within a specified range.

Commented [WU11]: timeOfNext move is a variable that holds the second which we want to wait for until the CPU will make another move.

Commented [WU12]: Iterate through all of the cards on the table and set them so that they are not clicked on and not selected by the player. This is basically just doing a clean up so as to make sure everything is clear for the CPU to make a move.

Commented [WU13]: Now call the cheat method to fill up the cheat list



Future Improvements

Save Game Option

Additional addons to the game would include the ability for the player to save his game allowing him to pick up his game where he left off.

Multiplayer

Local multiplayer functionality to allow players to compete with their friends.

Network Multiplayer

Functionality to allow people from all over the world to engage in head to head set battles.

Upgrades

Upgrades for player health, items to be able to freeze the CPU timer, show a set on screen, or do double damage.

Bonus Levels

Extra levels to earn in game currency to purchase upgrades.

Improve Code

Refractor and improve upon the code to be more efficient and lightweight.

Multiplatform Devices

Port the game to be playable for iOS devices and android phones.

Improved Graphics

Improve the overall rendering of the game instead of using plain old raster images.