# Blacjack Design Document

Allan Lavell

February 8, 2008

# Introduction

Blacjack is a card game that has been around for several centuries, with the French usually being credited for its creation sometime in the 1800's. It is still widely played today, and is very popular in casinos around the globe. This program is a simulation of the card game, where the player can bet virtual money and test his skills against the computer. It has a simple interface, is easy to pick up and play, and the gameplay can go as quickly as you want it to.

# Rules of Blackjack

This section will outline the basic rules of Blackjack. There are many variations of the rules. This game includes many of the commonly accepted rules, with some exceptions.

## Basic Rules

The object of a game of blacjack is to get a hand of cards that has a value of 21 without going over 21 (which is called "busting"). The player plays against the dealer, who is also trying to get to 21 without busting. A round of blackjack normally goes like this:

- The 52 card deck is shuffled
- 2 cards are dealt to the player and 2 cards are dealt to the dealer
- One of the dealer's cards is face-down and the other card is face-up
- The cards have a "value" equal to the number printed on a card, regardless of set. Kings, Queens and Jacks are all worth 10, and an Ace is worth 11, unless the player would bust, in which case the Ace is worth 1
- If the player gets a card with a value of 10 and an ace, he has blackjack, and has automatically won the round, unless the dealer also has blacjack, in which case the round is a push (tie). If a player gets blackjack, his bet is paid back 3:2.
- Otherwise, the player can hit, stand, split, or double. *Note: The split function is not available in the current version of this game*
  - **Hit**: When a player hits, he is handed a new card from the deck.
  - **Stand**: The player doesn't take any more cards.
  - **Double**: The player takes ones more card, doubles his bet, and stands [Can only be used if the player has 2 cards in his hand)
- If the player stands and hasn't busted, the dealer has to hit until he has a value of 17 or over, then stands. The values of the two hands are compared, and the higher value wins.

# Resources Used

## Programming Language

### Python

http://python.org
*From the website:* Python is a dynamic object-oriented programming language that can be used for many kinds of software development. It offers strong support for integration with other languages and tools, comes with extensive standard libraries, and can be learned in a few days. Many Python programmers report substantial productivity gains and feel the language encourages the development of higher quality, more maintainable code.

## Algorithms

### Fisher-Yates Shuffling Algorithm

http://en.wikipedia.org/wiki/Knuth_shuffle
I implemented this algorithm in Python to shuffle the deck. It is a well-tested and accepted algorithm for non-biased shuffling. Here is how it works:

1. Let A1 := 1, A2 := 2 and so on up to AN := N, and let n := N.
2. Pick a random number k between 1 and n inclusive.
3. If $k \neq n$*, swap the values of Ak and An.
4. Decrease n by one.
5. Repeat from step 2 until n is less than 2.

* As mentioned in the wikipedia article, checking for when $k \neq n$ is an unnecessary step, because swapping the same card with itself has no net effect. I chose to leave this out of my program, based on the assumption that checking for that condition every time would be slower than just letting it happen occasionally. The speed difference is probably trivial anyways.

## Libraries

### Pygame

http://pygame.org
For all of the graphical aspects of the game, the popular Python game development library Pygame was used. Pygame has been designed to make game design for python a lot easier by supplying functions for creating a screen (the window that the user views), creating and displaying sprites (images with specific properties useful for drawing them to the screen), playing sounds, and many other things. It is an open-source project.

### random

This is one of the Standard Libraries included in Python. It has functions for randomization, shuffling, etc.

### sys

This is another of the Standard Libraries. The main function I used from it is sys.exit(), which cleanly quits the program.

### os

This is a Standard Library that deals with platform-specific code. It was used to make sure the resources would load across all platforms.

## Images

### Cards

http://www.eludication.org/playingcards.html
The playing card images used in the game were created by Jesse Fuchs and Tom Hart, and are released under the Creative Commons License. Here is a link to the license:
http://creativecommons.org/licenses/by-sa/2.5/

## Sounds

### Click

I just recorded myself clicking the mouse, and edited it in Audacity (a free audio editing program).

## Editing Programs

### GIMP

http://www.gimp.org
*From the website:* GIMP is an acronym for GNU Image Manipulation Program. It is a freely distributed program for such tasks as photo retouching, image composition and image authoring.

It has many capabilities. It can be used as a simple paint program, an expert quality photo retouching program, an online batch processing system, a mass production image renderer, an image format converter, etc.

GIMP is expandable and extensible. It is designed to be augmented with plug-ins and extensions to do just about anything. The advanced scripting interface allows everything

from the simplest task to the most complex image manipulation procedures to be easily scripted.

GIMP is written and developed under X11 on UNIX platforms. But basically the same code also runs on MS Windows and Mac OS X.

### Geany

http://geany.uvena.de/
*From the Website:* Geany is a text editor using the GTK2 toolkit with basic features of an integrated development environment. It was developed to provide a small and fast IDE, which has only a few dependencies from other packages. It supports many filetypes and has some nice features.

## The Code

The project started out as a text-only version of the game Blackjack which I wrote in a few hours after learning the basics of the game. I then took a lot of the basic functions from the original text version and started the graphical version of the game. I tried to separate functions that handle basic blacjack game logic as much as I could from classes and functions that handle the graphics part of the program, but there is some overlap. The basic descriptions have been pulled from the game code, where I documented as I went along, but have been elaborated for further explanation of the code.

### Initialization Functions

- **imageLoad:** Function for loading an image. Makes sure the game is compatible across multiple OS'es, as it uses the os.path.join function to get the full filename. It then tries to load the image using the pygame.image.load function, and raises an exception if it can't, so the user will know specifically what's going if the image loading does not work.

- **soundLoad:** Function for loading a sound. Makes sure the game is compatible across multiple OS'es, as it uses the os.path.join function to get the full filename. It then tries to load the sound using the pygame.mixer.Sound function, and raises an exception if it can't, so the user will know specifically what's going if the sound loading does not work.

- **display:** Displays text at the bottom of the screen, informing the player of what is going on. It uses the pygame.font module to render the text to the screen.

- **playClick:** Loads and plays the clicking sound. Is called whenever the user clicks a button.

## Game Logic Functions

- **gameOver:** Displays a game over screen in its own little loop. It is called when it has been determined that the player's funds have run out. All the player can do from this screen is exit the game.

- **shuffle:** Shuffles the deck using an implementation of the Fisher-Yates shuffling algorithm. n is equal to the length of the deck - 1 (because accessing lists starts at 0 instead of 1). While n is greater than 0, a random number k between 0 and n is generated, and the card in the deck that is represented by the offset n is swapped with the card in the deck represented by the offset k. n is then decreased by 1, and the loop continues.

  The python random module also has a built-in shuffle function, which works very well too. I ran tests on both my shuffling algorithm and the random module's algorithm, and they yielded similar results. The tests can be found in the 'testing' folder. I wrote my implementation algorithm mainly to point out that I am aware of the fact that a bad shuffling algorithm can have a major impact on a card game, and would especially dangerous if used in a commercial card game where real money was at stake.

- **createDeck:** Creates a default deck which contains all 52 cards and returns it. The deck in this game was set up as a list. The naming scheme (consistent for the code as well as the actual filenames of the graphics) works like this: the first character is the first letter of the set ("d" for diamonds, for example); the second charcter is the value of the card (e.g 8, or "q" for queen). In the creation of the deck, I started off with a base list of "special" names (not numerical, ex: "dq" - Queen of Diamonds), then looped through 2 to 10, assigning each number a set and adding it to the deck. The deck is then shuffled and returned.

- **deckDeal:** Checks to see if the deck is gone, in which case it takes the cards fromthe dead deck (cards that have been played and discarded) and puts them in the live deck. Shuffles the deck, takes the top 4 cards off the deck, appends them to the player's and dealer's hands, and returns the player's and dealer's hands.

- **hit:** Checks to see if the deck is gone, in which case it takes the cards fromthe dead deck (cards that have been played and discarded) and shuffles them in. Then if the player is hitting, it gives a card to the player, or if the dealer is hitting, gives one to the dealer.

- **checkValue:** Checks the value of the cards in the player's or dealer's hand. It starts off by setting up a variable, totalValue, with the value of 0. This variable will be used to measure the value of the player's hand. The function then loops through the cards in the hand which has been passed to it, checks the name of the card, and adds the numerical value equal to the value of the card to totalValue. If the hand is then determined to have a value of over 21 after all the cards have been added up, the

hand is checked for aces, and if there are any aces their value is diminished from 11 in accordance to the rules of blackjack. In situations where there are multiple aces, only the necessary number of aces have their value changed to get the totalValue under or equal to 21.

- **blackJack:** Called when the player or the dealer is determined to have blackjack. Hands are compared to determine the outcome. There are three possibilities:
  - Player and dealer both have blackjack, in which case it's a push
  - Player has blackjack and dealer doesn't, so player wins and has his bet paid back 3:2
  - Dealer has blackjack and player doesn't, in which case player loses

  The blackJack function writes a message on the bottom of the screen according to the outcome by calling the display function (discussed later). It then calls the endRound function (also discussed later).

- **bust:** This is only called when player busts by drawing too many cards. This function just displays the "You bust!" text and calls the endRound function.

- **endRound:** Called at the end of a round to determine what happens to the cards, the money gained or lost, and such. It also shows the dealer's hand to the player, by deleting the old sprites redrawing them with the first one face-up. If the player has run out of funds, it calls the gameOver function.

- **compareHands:** Called at the end of a round (after the player stands), or at the beginning of a round if the player or dealer has blackjack. This is the function that makes the dealer hit if he has a value of less than 17. It then compares the values of the respective hands of the player and the dealer and determines who wins the round based on the rules of blacjack.

## Sprite Classes and Methods

- **cardSprite:** Sprite that displays a specific card. Since the filenames of the card images all correspond to the names used in the program for the cards, it is easy to load the images.
  - When the card sprite is initialized, it calls the pygame.sprite.init, which sets up the initial attributes of the sprite. It sets its self.image attribute to the corresponding image of the card, which is loaded through the function imageLoad.
  - The sprite's update method (which is called every frame) simply sets the card's position as the center of the rectangle that contains the image. Since the card won't be moving around the screen, no other code is required here.

- **hitButton:** Button that allows player to hit (take another card from the deck).
  - The init method calls the pygame.sprite.init, which sets up the initial attributes of the sprite. The self.image attribute is set to "hit-grey.png", which is the hit button but greyed out.
  - The update method of the card If the button is clicked and the round is NOT over, Hits the player with a new card from the deck. It then creates a sprite for the card and displays it. Also, if the round is over, the button is displayed as grey, because the player can't hit when the round is over.
- **standButton:** Button that allows the player to stand (not take any more cards). The methods for drawing the sprite of this function are similar to those of the hitButton.
- **doubleButton:** Button that allows player to double (double the bet, take one more card, then stand).
- **dealButton:** A button on the right hand side of the screen that can be clicked at the end of a round to deal a new hand of cards and continue the game.
  - If the mouse position collides with the button, and the mouse is clicking, and roundEnd does not = 0, then Calls deckDeal to deal a hand to the player and a hand to the dealer. It then takes the cards from the player's hand and the dealer's hand and creates sprites for them, placing them on the visible table. The deal button can only be pushed after the round has ended and a winner has been declared.
- **betButtonUp:** Button that allows player to increase his bet (in between hands only).
- **betButtonDown:** Button that allows player to decrease his bet (in between hands only).

## Main Game

In between the declarations of the sprites and the beginning of the loop of the game, there are several variables declared. They will all be accessed later by the program in the game loop and modified, but they need to be declared before the loop starts to initialize things.

The while 1 loop, which contains the code for the handling of the game, starts. It does things like blit (draws) the background image to the screen, checks if the player has no funds left (in which case it calls gameOver), checks for user input via the pygame event queue (it loops through the event queue, and does different things based on which button is being pressed), checks if the player or dealer has blackjack, updates the sprites on the screen and gets their return values, draws the sprites to the screen, and then loops again. Each cycle in the loop represents one frame displayed. Since there is very little animation in the game (read: none), the framerate should not be an issue and the game should run very well even on older hardware.

# Testing

This game was mostly tested by me, but my brother Chris Lavell and my friend Alex Moriarty also helped find a few bugs. There was no real "testing procedure", other than playing the game and seeing what worked and what didn't. Playing it a lot helped, because there were some bugs that were not apparent at first, but were noticeable if you were paying attention. Consider this situation:

- Player's funds = $50
- Player's bet = $40
- Player plays a hand and loses, therefore losing $40 and putting his funds down to $10
- At first, there was no code to stop the bet from remaining at $40, so he could still bet more money than he had

This bug was only evident after playing for a while. I do realize that sometimes in real life, people bet money that they don't actually have, but I wanted to keep the game simple and arcade-like, so you can't win your way back out of the hole. Zero dollars is zero dollars.

I wanted to make sure that the cards were being shuffled evenly, so I wrote a script to test the shuffling algroithm included in the random module and one to test the shuffling algorithm that I wrote. The scripts create a list with three values (1, 2 and 3), and shuffle the list using their respective algorithms. The scripts then check what the combination of the list is (for ex: [1,3,2] or [3,1,2]) and add +1 to a counter that is keeping track of how many times that particular combination has been created. The process is repeated this process 600 000 times, then the results are printed. The 6 different possible combinations all have approximately 100 000 counts in both scripts. Therefore, both shuffling algorithms are doing a good job, and are not over-representing one possibility.

The scripts are included in the "testing" folder, and they're called pyrand.py and fishtest.py.

# Conclusion

This project has been a good learning experience. It has been fun to make, and it is easy to play. I learned many lessons about programming a larger project, and I have a better understanding of how to write a game with python and pygame. I also learned how quickly you can lose a hundred bucks at a casino!