Technical Manual Canoga C++

Devinder Sodhi

# BUG REPORT

Several bugs were noted, introduced and fixed over the course of the game. One example was.

start a new game. have two rolls of snake eyes.

 end turn.(you have covered your number 2 and are at an invalid turn) save the game.

Load game. There is a check for win at beginning of game.

this method of checking for win has no means of knowing what the last move was made by the player (cover). it can only assume win states by looking at board states if its going to declare wins.

**Result**: false Assumption that player won by Uncovering last round.

**Proposed Fix**1: add a variable to the save file that defines the "last move that was made" ie Cover or Uncover. Make use of this variable on game fix time.

**Proposed Fix2**: without affecting the save file (this is the third time i m suggesting this) Do not allow saves or wins for the entirety of the first turn. This Eliminates several bugs your proposed implementation has Including getting rid of most chances of a false win.

## The majority of the bugs of the design that were found were addressed. The current bugs include some issues with the way things are implemented:

* Invalid file name entries are tested, so are incorrect formats of files - but some cases are ill-defined and just close out.
* incorrect save file name results in a second option which then lets you enter the name again, but forces a second space. Probably an ill placed cin.ignore.
* A game only allows INTMAX turns.

# FEATURE REPORT

* ALL REQUIRED FEATURES ARE MET. no missing features
* As the Rubric will mention:
* Can play a game, alternate, go through the hoops and what not.
* Option to play 9-11 tiles
* Working AI with a strategy.
* visual interface.
* Save/Load functionality
* Ability to load dice.
* Computer has a strategy. Will favor maximizing score.
* helper functions to simplify data validation

# Classses/Data Structures.

## Data Structures:

* Vectors: used to hold tilesets of players.
* Vectors of vectors: used to support more such functionality
* Deque -> to hold Dice rolls. Game Logic will prevent unpredictable dice throws from file.
* A tree -> to hold the numbers that ad up to a target value. A tree will hold up in sorted order subsets that add up to ten exactly. Any larger values will be pruned. Any values not approached will be ignored. 1 might be a root. and have the immediate branches 2,3,4,7. The case of 2 will have 3,4,7, the case of three will have 4 and 7, of These two cases 1-2-3-4 is 10 and 1-2-3-7 goes over the limit. If any combination goes over 10, it gets pruned off.
  + example: i have digits 1,2,3,4 and 7 that are available.
* PlayerSquare -> a Class to help represent each player square on a tile

## Inheritance:

Human and Computer Class inherit their strategies, and some common functionality from the Player Class. Furthermore the MakeMove() functionality differs in each child class but is pointer accessible polymorphically from the Parent.

# Classes

## Tournament:

* Controls the meta of the rounds.
* Allows Newgame/loadgame/continue game scenarios
* Triggers Game Class to make those scenarios possible
* Reads out total scores at the end

## Game:

* The game flow is controlled here. Allows control of the board and playing of rounds.
* Load/Save are handled here
* Round initialiazation is handled here
* Triggers players/Board/BoardView
  + Hands off pointers as part of
* manipulation and determination of the current player is handled here
* winner is determined here.
* triggers boardview to show current game state.

## Player

* generic player properties and methods are here, including strategy which the human gets help from and the computer employs
* Allows self identification as human or computer
* setting of score is done here
* Checking for Valid moves (really through the board) is handled here

## Human

* Child of player.
* has its own version of virtual function MakeMove
* has additional interface to ask human what they want to do as opposed to computer which makes the moves itself
* takes a pointer to the board.

## Computer

* Employs Player strategy
* Makes a move polymorphically, doesn't have the userinterface of human

## Board

* Holds game state.
* settings, getting of all board variables. including tilesets and pointers to them
* calculation of valid moves
* determines the vector of interest automatically by looking at next player and the intention of the player to cover or uncover

## Boardview

* Using a pointer, determines game state and draws the current Game state. Thats it

# LOG

**September 4**

Outlined the requirements in a notebook. Setup a save file and a github repository and started the project file.

2hrs

wrote out the skeletons to the classes. Tried to wrap my head around the structure.

1hr

**September5**

sketched out the first part of what i'll be doing. Organized a hierarchy for Tournament, Game, Board Classes and assigned member functions

2 hrs

Decided its good to have classes setup, but need to work from groundup. Consequently shifted focus to Board Class to get the basic game started.

Need a game Start state. Gave up on this after 1 hour

1 hr

Got back to Tournament Class and set up the Tournament::GameOptions(). This function will allow input to be properly assesed to start a new game or load a game.

Researched and implemented proper input validation

30 min

Restructured Tournament Class to have a cleaner flow conceptually.

**September 9**

Wrote a pointered version of my C++ Game Class. Broke it down after realizing that I could keep the class in scope, without adding the complexity. http://stackoverflow.com/questions/22146094/why-should-i-use-a-pointer-rather-than-the-object-itself

5 hrs.

**September 10**

Decided to go back to a simple implementation. Started with eliminating a lot of variables to pass data between classes.

Moved load game data from tournament to Game Class. Started writing getters and Setters for teh board class.

ironed out some reference bugs, test code so far.

reading in file info is next.

**September 11**

wrote out the Load file function. Adjusted many small bugs

2 hours

**September 12,**

enabled Board to be initialized. Created the Initializer function in Game

2hrs

**September 13**

Wroteout the new game functionality. Created Players/human/computer classes and roll dice fucntionality that will conform to requirements of

polymorphism. Read a chapter on the subject of polymorphism

5hrs

**September 15.**

Confirmed code so far is tested to be error free. started visualizing continue game scenario where advantages are taken account of

2 hrs

**September 17**

probably should make "CheckLoadFilesAndPAss" a bit more independant and re-usable for the dice roll down the line.

removed the continuegame function

1 hr

removed the playGameROundsfunction and put most of the relevant logic in GameOptions. Made more sense -> prevented a twisted Stack by two functions consitently calling each other.

Added logic for getting the winning score from the board.

think about about where I'll be adding the winning score.. DeliverAdvantage() now runs and works from the Board.

1 hr

Decided to drop some BOard functionality into its related view class

Realized I need a constructor list to get the BoardView to trigger the correct Constructor...

1hr

**September 18**

Wrote the display function in BoardView;

Setup the RoundFunction -> ie it recognizes current player and switches players for next round.

2 hrs

**September19**

Game Class now implements load and new games more fluidly. Changed "start" and "continue" to be run with a bool different in the initialization.

Changes how variables are set.

2hrs

Board intelligently recognizes user given next player and move intended and hands off the correct reference to tileset

2 hrs

**September 21**

Board view functions are complete.

Loading a game is now possible

2hrs

**September 20**

Human Class has Make Move. A move can be made

2hrs

Player Class works as expected and polymorphism is implemented

2hrs

**September21**

Ai strategy is complete. It maximizes score and goes after the "big wins"

2 hrs

Debugged through memory leaks after attemping first run

The memory leaks were mostly related to pointer traversal and bad iteration counting.

2/3/2 hrs

**September 22**

Dice File now loads

20 minutes

Now its smart (Dice file reader)

40 min

Turns are now accounting for some of the bugs that were encountered in design.

4 hours

September 23

Kumar says my game play flow is incorrect.

Rewiritng game class to meet the requirement:

2/2/2

**September 24**

Finished rewrite. Game now follows the flow

1. check win

2. make turn happen

3. check win within turn

4. allow save if conditions are met

3 hrs

**September 25**

Documented

5 hours (I tried to be careful)



