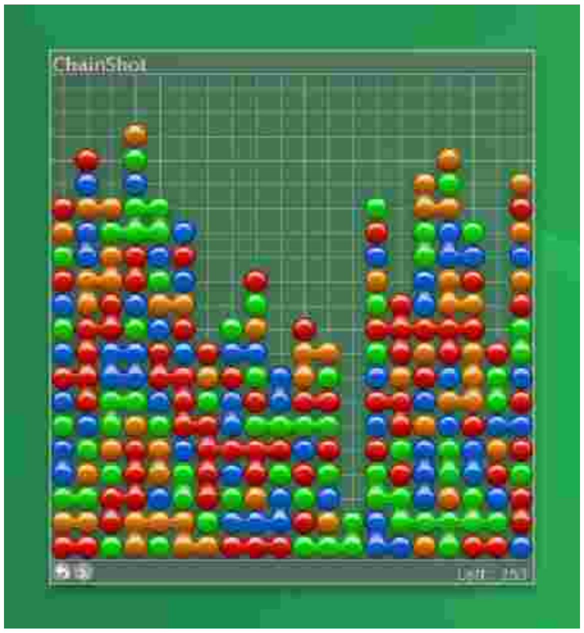
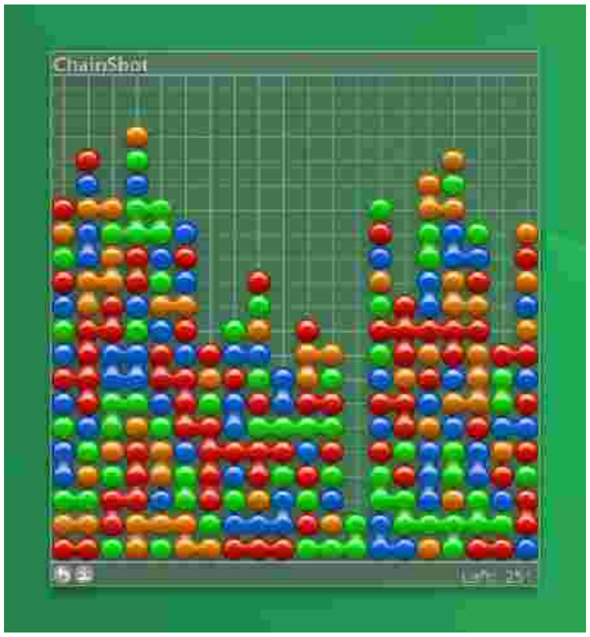
Term Project
Fall 2012

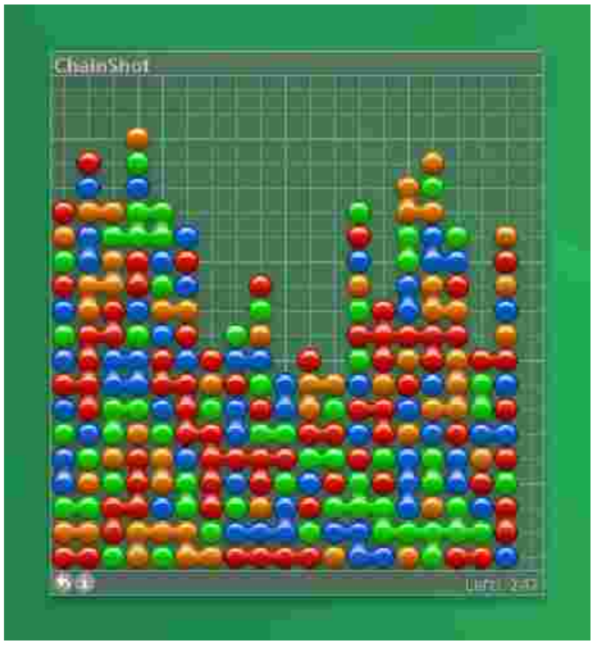
This semester you will implement a game called "ChainShot". You can find a version of this game at: http://www.oopixel.com/games/samegame/. This game is played on a 5x5 to 25x25 grid where each square is initially occupied by a colored bead. To refer to positions on the grid, we use the following numbering convention: each position is referenced by two numbers: its row and its column. In the gird below, the lower left corner is position (1,1), the lower right corner is (1,20), and the upper left corner is (20,1).

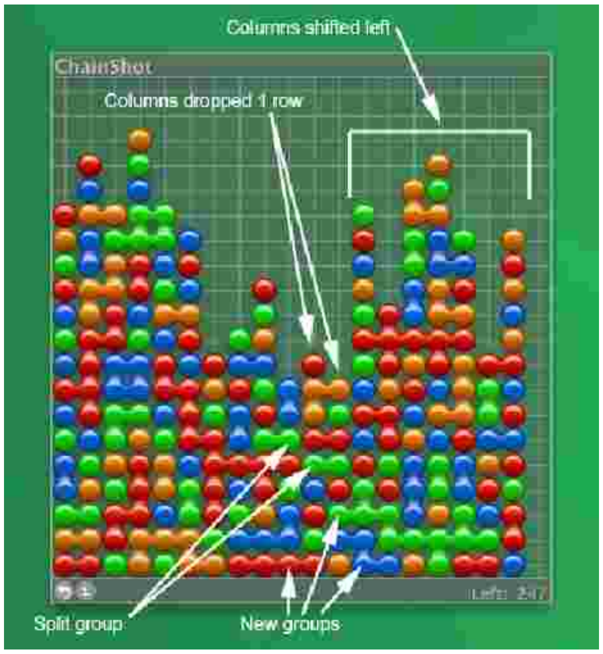


Beads of the same color that touch each other vertically or horizontally (but NOT diagonally) are considered a group. Any group can be removed from the board during a move while single beads cannot. When a group is removed, all of the beads supported by the group drop down as far as possible in their respective columns. The dropped beads can possible breakup old groups or form new groups in their new positions. In the right most column of the example to the left, removing the vertical group of two green beads, positions (9,20) and (10,20), will cause the six supported beads to drop. The red bead on the bottom now in (9,20) will form a group with the red bead in the column to the left, (9,19), as shown in the figure below.



If a column becomes empty, every column to the right shifts to the left to fill the empty column. In the figure to the left, column 13 contains 2 green beads, (1,13) and (2,13), connected with two other green beads in columns 11, (1,11), and 12, (1,12). If we remove this group, the remaining beads in columns 11 and 12 will fall one position and all of the beans in columns 14 through 20 will shift one column to the left. New groups are then formed where similar colored beads match between columns 10 and 11 and the old columns 12 and 14 which are now adjacent, and column 20 will be empty, as shown on the next page.





A game starts with the entire grid filled with beads. The objective of this game is to remove as many beads as possible – hopefully, every bead –from the grid while obtaining the highest score. Scoring of any move is computed as $(n-2)^2$ where "n" is the number of beads removed. The best game score is determined by: the smallest number of beads left on the board, then the highest sum of the individual moves. So, the following games scores are ranked from best to worst:

Beads Left	**Move Score**
1	21,450
1	20,100
2	30,423
2	25,678
2	18,927
3	24,567

Since you will not want to waste time on developing graphics for this game, you program will use a simple matrix of characters to represent the beads in the grid. The grid size can vary from 5x5 to 25x25. The number of colors for the beads can vary from 3 to 5. The letters used for each color are: R – red, G – green, B – blue, L – Black, and O – orange. A move is specified by two numbers representing the position of *any* bead within a group, so if we were to want to remove the red bead group in position (1,1) and (1,2) in the above game, we could specify *either* bead.

All students are required to work individually. CAP5535 are required to use LISP in developing their solution. CAP 4621 students can use the language of their choice (though a 10% bonus will be given for using LISP).

Project Time Table

The following are important dates when project reports or code submissions are required.

Oct. 22 Initial submission of project

This submission will show that your program can play this game with no intelligence. Your program should request and read the name of a file containing a set of data describing pieces on the board, should allow a human to specify moves (moves will be specified as two numbers - the first is the row and the second is the column - separated by a blank with no other punctuation), should make the move and display the new game board, should recognize when a game is completed (no beads left on the board or no adjacent beads of the same color remain), should display all moves made in playing the game (these can be displayed individually with each move), and should ask if another game is desired (repeating this process if required). Any other added features ("bells and whistles") which you feel should be part of the user interface should be demonstrated here (for example, displaying a set of rules to the game is *highly* recommended). Each student will turn in a zipped copy of their program on this date with a "read me" file describing how to execute your program. The main routine for your program should be called "chainshot".

Oct. 26 Written report

A PDF (preferred), Open Office, or Word file of your written report should be turned in on this date (1-2 pages in length, single spaced). This report will detail what intelligence you intend to implement in your program. Note: you might not be able (due to time until the end of the semester, time limits for a move, and space within the machine) to fully implement all of this. Your report should therefore be structured to contain the following two sections:

1. This is what I will definitely implement, and
2. This is what I hope to implement provided I do not have any major difficulties.

Nov. 4 Search Implemented

On this date I encourage you to have a search program implemented that, as a minimum, returns a legal move for a given board position or identifies that no move can be made in that position. ***No submission of code*** is required on this date!! ***This is merely a road marker for you.*** From this date until the final submission, you can concentrate on implementing your intelligence.

Nov. 30 All projects must be fully implemented and running for final submission

This submission will be used to show what you have as a final program. You will be graded on the program playing a legal game (not making illegal moves), any changes which have been made since the last demonstration, and how well the program plays. Your program should allow humans to play manually or the computer to play automatically. During automatic play, your program should display the board after each move, and should print a list of moves made after the game is completed with the total CPU time required to generate that solution and number of beads remaining on the board. You are to submit a zipped version of your program and, again, a read-me file describing how to load your program and any other directions required to execute your program.

Nov. 30 – Dec. 4 Tournament

I will post a series of data sets that you will be required to run your program against (ranging from very easy to extremely hard). You must report the execution (CPU) time taken to generate each solution, the number of beads remaining on the board, and the score obtained. Each solution must be found in a total of no more than 30 CPU seconds (I reserve the right to change this value).

We will verify your results by testing your program ourselves on one or more of the tournament data sets to verify your results.

Dec. 5 Final written report

This report will detail the entire project. As a minimum this report should include:

1. A description of the game which was implemented.
2. A description of the implementation approach which was taken.
3. A description of all programs/procedures/methods written and all major variables used.
4. A flowchart of the program.
5. A calling hierarchy and/or UML diagram showing the various methods/functions and how the various they call each other.
6. A description of what intelligence was implemented and what was not. Discuss why the non-implemented intelligence was not implemented.
7. A discussion of what you would do differently if you were able to start over.
8. The time used, number of beads remaining, and the score for each solution that was generated by your program for each tournament data set.

Grading

The grading of the project will be according to the following scale:

Initial Demonstration 10%
Intelligence Report 15%
Play against tournament data sets 25%: 15% for playing all data sets and reporting your results and 10% for the intelligence of play.
Final Demonstration 15%
Final Report 35%

Sample Data Sets

The following data sets are given for you to use in learning how to play and to build your skill level. They range from easy to difficult. I do not know if each of them can be solved!

```
1. GBOOB
   BGGGO
   GBOBB
   BBBGG
   OBBBG
```

```
2. GGBGG
   BBOGO
   GBOBO
   GGOGO
   BGBGG
```

```
3. BBBBGGB
   GGBBGBB
   OOOGBBO
   GGOGBOG
   GOOOOGG
   BOOGGOO
   OOGBOBB
```

```
4. RRGGRRG
   OBBOOGO
   GBBRBOR
   GBBORGG
   RRBBBOR
   BRROOBG
   BRGBRRB
```

```
5. LGROLRBOGB
   LLBRLOGLRG
   GGORGBBLLL
   BLLBOORRRB
   GROBOBBLGO
   BOBRLLGBBB
   BOOOBBRGOO
   GRRROOGROG
   OLBLOLBROB
   BBBLROBOGO
```

```
6. GGOGRLRLRL
   LOOOLRBOOL
   BLOLOBLGBO
   GBOGLLLBLG
   RGBBGBORBL
   LGGRGBRGGB
   OOROROBRLB
   GGOGBRBROL
   ORLGLGBBOG
   GRBBOLROGG
```