Programming for Science
Workshop 5

The aim of this workshop is to give you some practice in decomposing problems and thinking about algorithms and functions to solve them. You should now be familiar with the main elements of common languages (variables, lists, loops, conditionals and functions), so you know what sort of tools are at your disposal. However, there is not supposed to be much (if any) programming.

Depending on when you do this workshop, the new CA will either have just been set or is imminent, so please make sure that you have caught up with any outstanding workshop exercises **before** doing any programming for this one.

Do this workshop in groups of three or four, using paper and pen rather than computers! Many people working on the first CA started sitting in front of a computer. It's really much more effective to decompose the problem with pencil and paper. Have a good idea of how the program is going to work and what the functions are before you start to program it.

1 Problem decomposition

For each of the following problems, decide on what the functional units are in the program. Decide how you might represent the data in the program and thus what each function would do. It may be helpful to write pseudo-code or flow charts to show how your functions fit together. How might you test each function? Can you think of simplifications of the problem or components of the problem that would allow an incremental development?

If you can decompose the problem far enough write doc-strings for the functions describing what each does, what its arguments are and what it returns. What other functions does your function need to call?

- A drunkard steps out of the pub and each second takes a single step along the street either to the left or to the right. Your program should simulate the drunkard's walk and find out how far on average he is likely to be from the pub after, say, 10000 steps. (This week we will see a `random` function that returns a random number between 0 and 1.)
- A computer program to simulate a jukebox. Assume that a function is available that will read a song from a file as a `song` variable, and another function `play(song)` that will play the track. Assume that you have a function that will ask the user for the name of a song and a return a string containing whatever the user has typed.
- A program to play the word game hangman. Assume that you have a function that will ask the player for a word or a letter. Your program should use the `TurtleGraphics`.

  (If you have time, this is fun to program and will be the subject of a future workshop.)
- The games Snakes and Ladders. This is the subject of lectures this week and next and well worth thinking about.
- The game Beggar your Neighbour. If you don't remember the rules, here's what wikipedia says:

> A standard 52-card deck is divided equally between two players, and the two stacks of cards are placed on the table face down. The first player lays down his top card face up, and the opponent plays his top card on it, and this goes on alternately as long as no ace or face card (King, Queen, or Jack) appears.
>
> If either player turns up such a card, his opponent has to pay a penalty: four cards for an ace, three for a King, two for a Queen, or one for a Jack. When he has done so, the player of the penalty card wins the hand, takes all the cards in the pile and places them under his pack. The game continues in the same fashion, the winner having the advantage of placing the first card. However, if the second player turns up another ace or face card in the course of paying to the original penalty card, his payment ceases and the first player must pay to this new card. This changing of penalization can continue indefinitely. The hand is lost by the player who, in playing his penalty, turns up neither an ace nor a face card. Then, his opponent acquires all of the cards in the pile. When a single player has all of the cards in the deck in his stack, he has won.
>
> `http://en.wikipedia.org/wiki/Beggar-My-Neighbour`

Think hard here about:

- the deck of cards: do you need to represent the suits?
- how will you represent the cards that are held by the players and the ones on the table?
- how does the game end?

This is quite difficult to program (it was almost a CA question, but would take too long), but it is well worth while thinking about how to decompose it.

- A program to keep track of books in a small lending library. Assume that the names of the books the library has can be read from a file and initially all the books are in the library. This could be quite complicated, but make it as simple as you can.
- You are given the following information.
  - 1 Jan 1900 was a Monday.
  - Thirty days has September, April, June and November.
    All the rest have thirty-one,
    Saving February alone,
    Which has twenty-eight, rain or shine.
    And on leap years, twenty-nine.
  - A leap year occurs on any year evenly divisible by 4, but not on a century unless it is divisible by 400.

  A program to computer how many Sundays fell on the first of the month during the twentieth century (1 Jan 1901 to 31 Dec 2000)?

2 Yahtzee

Yahtzee is a dice game that was first marketed in 1956. The winner of the the game is the player who scores the most points from rolling five die to make different combinations. There are thirteen different combinations as shown in figure 1, which shows a score card from the game.

On a go, a player can roll the five dice up to three times to try and find the combination that will give them the best score for that go; each combination scores a certain amount of points, some have fixed values and others have the cumulative value of the dice (see score card). Each Yahtzee game comprises thirteen rounds. For each round a player must select a combination to put a score in – once that combination has been chosen, a player cannot choose it again in the game. Some times a suitable combination may not be available and therefore a 0 must go in one of the remaining combinations. For example, a player may only have a full house yet to obtain (3 die of one value, 2 die of another value) but gets the values of 2,2,2,3,5. They would have to put a 0 in their full house. A Yahtzee is five-of-a-kind and holds the game's highest point value of 50.

The task is to simulate a computer playing one game of Yahtzee.

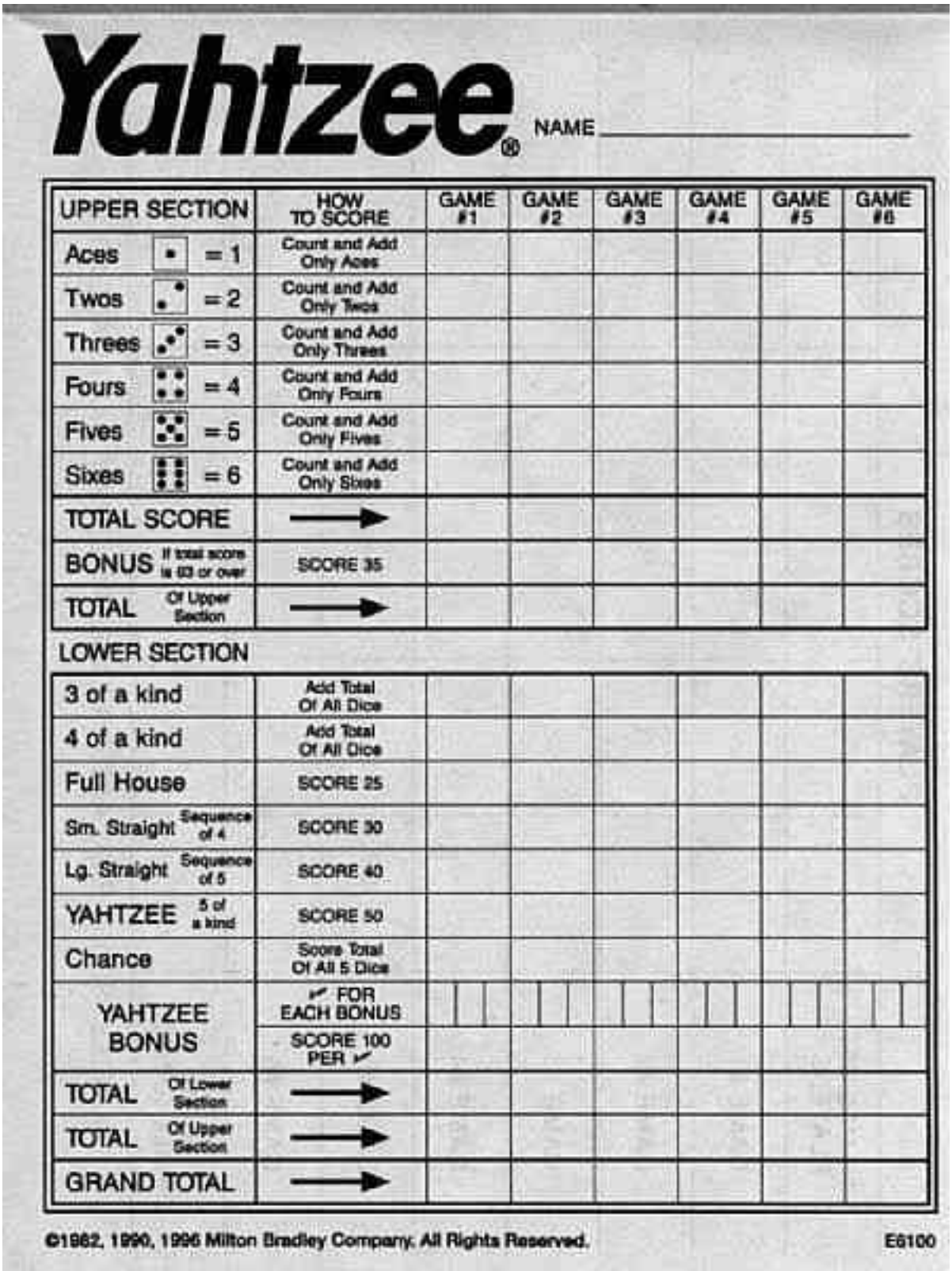
1. Break down the problem into manageable components (just like you have done for Snakes and Ladders in lectures).

2. Develop an algorithm for how you will identify, which combination that your five die should be chosen for each round. Remember each combination can only be chosen once.

3. Once you have broken your problem down, begin to program each component separately.

2.1 Notes

- You should think carefully about what you have already programmed in previous workshops.
- Think about what your approach would be when playing the game.
- Ensure you have each combination working before moving onto the next.



Yahtzee®

NAME _____

UPPER SECTION	HOW TO SCORE	GAME #1	GAME #2	GAME #3	GAME #4	GAME #5	GAME #6
Aces = 1	Count and Add Only Aces						
Twos = 2	Count and Add Only Twos						
Threes = 3	Count and Add Only Threes						
Fours = 4	Count and Add Only Fours						
Fives = 5	Count and Add Only Fives						
Sixes = 6	Count and Add Only Sixes						
TOTAL SCORE	→						
BONUS If total score is 63 or over	SCORE 35						
TOTAL Of Upper Section	→						
LOWER SECTION							
3 of a kind	Add Total Of All Dice						
4 of a kind	Add Total Of All Dice						
Full House	SCORE 25						
Sm. Straight Sequence of 4	SCORE 30						
Lg. Straight Sequence of 5	SCORE 40						
YAHTZEE 5 of a kind	SCORE 50						
Chance	Score Total Of All 5 Dice						
YAHTZEE BONUS	✓ FOR EACH BONUS						
	SCORE 100 PER ✓						
TOTAL Of Lower Section	→						
TOTAL Of Upper Section	→						
GRAND TOTAL	→						

Figure 1: A Yahtzee score card