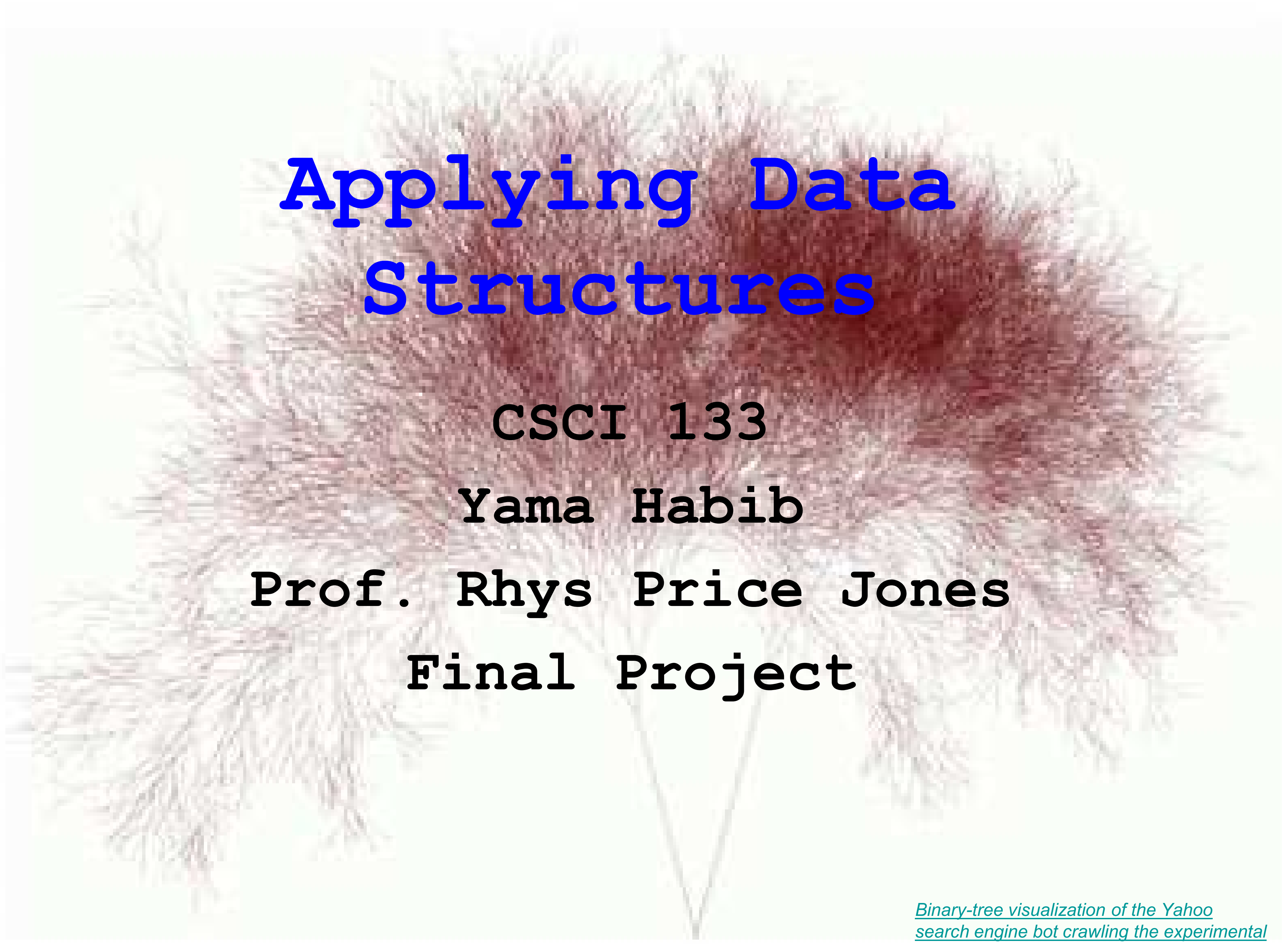


Applying Data Structures

CSCI 133

Yama Habib

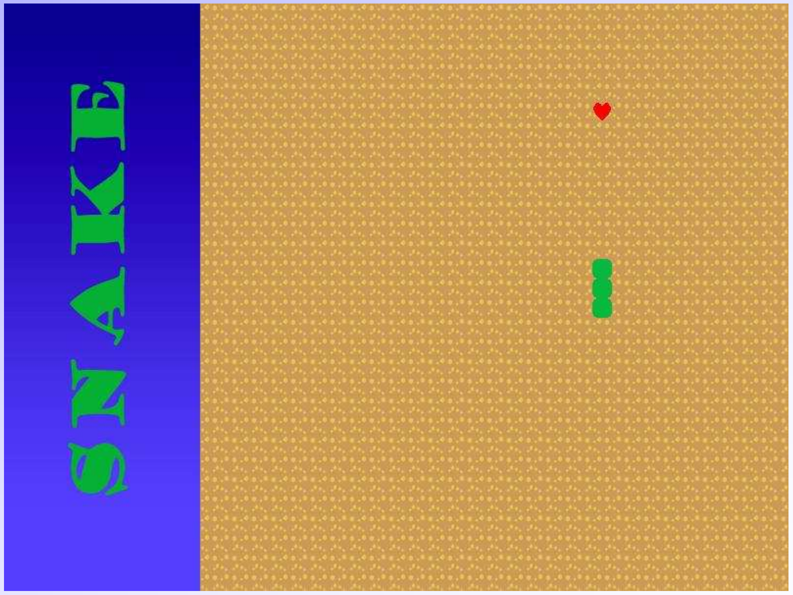
Prof. Rhys Price Jones

Final Project

Binary-tree visualization of the Yahoo search engine bot crawling the experimental website.

My Implementation

- "Snake" – A popular minigame where the object is to "eat" randomly appearing objects and grow without touching walls or other elements of the Snake.
- Implemented as a grid of icons whose values can be "blank", "snake", or "heart"



Basic Rules

- If “snake” icon and “heart” icon overlap, the “snake” icon shows and the Snake ‘grows’ by one unit.
- If a “snake” icon and another “snake” icon overlap, game over.
- “heart” icons can only appear on currently “blank” icons.
- If the a “snake” icon wants to be drawn off the grid, game over.
- If all icons are “snake” icons (unlikely!) game won.

Basic Rules (In pseudo-java)

- If “snake” icon and “heart” icon overlap, the “snake” icon shows and the Snake ‘grows’ by one unit.

```
if(grid[x][y] == heart)
{
  grow();
  grid[x][y] = snake;
}
else
  move();
```

Basic Rules (In pseudo-java)

- If a "snake" icon and another "snake" icon overlap, game over.

```
if(grid[x][y] == snake)
{
  gameOver();
}
```

Basic Rules (In pseudo-java)

- “heart” icons can only appear on currently “blank” icons.

```
void setHeart(int x, int y)
{
   if(grid[x][y] != blank)
   {
       setHeart(random(x), random(y)); // recursive!
   }
   else
       grid[x][y] = heart;
}
```

- Rather tedious, isn’t it? What happens if there aren’t a lot of blank tiles left??

Basic Rules (In pseudo-java)

- If the a “snake” icon wants to be drawn off the grid, game over.

```
if(x<0 || y<0 || x>SIZE || y>SIZE)
{
  gameOver();
}
```

Basic Rules (In pseudo-java)

- If all icons are “snake” icons (unlikely!) game won.

```
while(x<SIZE && keepSearching)
{
   while(y<SIZE && keepSearching)
   {
        gameWon = true;
        if(grid[x][y] != snake)
        {
                gameWon = false;
                keepSearching = false;
        }
        else y++;
   }
   x++;
}
```

- This looks particularly nasty! Don’t even read it!!

Basic Rules (In pseudo-java)

- As Rhys would say…

```
// the rest is GUI stuff.
```

- “You can do this in your sleep.”
- …That’s how Rhys can write our labs and still come to class well rested. ;)

What Data Structures Do We Need???

- The Snake – elements get added to the front, removed from the back.
- The Grid – Two dimentions, set number of elements
- HiScores? – Save a name and a number, sort from highest to lowest

What Data Structures Can Make Our Code Better?

- Inefficient searching for random spaces for hearts…
- *REALLY* inefficient method of checking if the game has been won…
- Can you guess which DS's we can use..?

Overview of Useful Data Structures

- **Array – Useful for set amounts of data with random-access**
- **Vector – Useful for (virtually) unlimited amounts of data with random-access**
- **Stack – Unlimited last in, first out data structure**
- **Queue – Unlimited first in, first out data structure**
- **Heap – Sorted list with efficient greatest/lowest-value-out functionality**
- **TreeSet – Alternative implementation of a heap with slightly less efficiency, but increased functionality**
- **HashSet – Extremely fast storage and access—but inefficient for sorting**
- **Does that help?**

Drum Roll Please…

- Queue – For the snake, that one's a no brainer!
- 2D Array – Perfect for a grid with easy coordinates
- Heap – Excellent for efficiently sorting and storing high scores
- 1D Array/Vector/HashSet/Stack – Doesn't really matter which, but a collection of readily randomized, currently unoccupied (blank) coordinates would up the efficiency of the code.
  - i.e. `if(choices.isEmpty()) gameWon();`
    - As opposed to the garbage in Slide 8…
  - Or: `setRandomHeart(choices.peek().getX(), choices.poll().getY());`
    - As opposed to the garbage in Slide 6…

Wanna try it out?

Click Here!