Outline

- Purpose
- Array Syntax
- ArrayLists
- Multi-Dimensional Arrays

Why Use Arrays? (1/2)
- So far, we've only studied variables that hold references to single objects
- But what about holding lots of data? Many programs need to keep track of hundreds/thousands of data instances
- Want to hold arbitrary number of objects with single reference – represents a collection of elements
  - allows for simple communication to multiple elements
- Arrays are the simplest data structure or collection - we'll also cover lists, queues, and stacks
Why Use Arrays? (2/2)

- Arrays allow instances of specific type to be "packaged" together and accessed as group
- What if there are 10 instances of Location
  - store all Locations in array for easy access (to film new seasons of Survivor: Rhode Island Season 2, CS15 Edition?)
- Arrays are ordered - helpful when wanting to store or access instances in particular order, e.g., alphabetically

Your lovely TAs

- We want access to all 39 UTAs!
  - Abigail, Adam, ..., Zahra
- Could use instance variables:
  
  ```java
  public class CS15TAs {
      private TA abigail, adam, ..., zahra;
  }
  ```
- Can't access 39 instance variables very easily
  
  - what if we wanted to access CS15 TAs from spring 2021, 2019, 2018, …

Arrays (1/4)

- Arrays store specified, constant number of data elements of the same type – our first homogeneous collection
  - each element must be same type or subclass of same type (polymorphism)
- Arrays are special in Java
  - special syntax to access array elements:
    
    ```java
    studentArray[index]
    ```
  - the index of array is always of type int
  - neither base type nor class, but Java construct
  - use new to initialize an array (even though it’s not a class!)
  - special syntax, does not invoke constructor like for a class
Arrays (2/4)
- Arrays only hold elements of specified type
  - when declaring arrays, state type of object it stores:
    - base type
    - class
    - sub-arrays (for multi-dimensional arrays – soon)
    - or for max polymorphic flexibility, interface or superclass
  - type can even be `java.lang.Object` to store any instance, but that isn’t useful: wouldn’t take advantage of compiler’s type-checking

Arrays (3/4)
- Every array element is an object reference, subarray, or base type. What real-world objects can be organized by arrays?
  - number of electoral votes by state
  - streets in Providence
  - Strings representing names or Banner IDs of people in a course
- Elements ordered sequentially by numerical index
  - in math, use subscript notation, i.e., $A_0, A_1, A_2, ..., A_{n-1}$
  - in Java, use index inside brackets, i.e., for an array of students:
    `students[0], students[1], ..., students[n-1]`

Arrays (4/4)
- Arrays store objects in numbered slots
  - for array of size n, first index is always 0, last index is always n-1
- Common graphical representations of arrays:

Note: 1-D Arrays are called vectors, and 2-D or n-D arrays are called matrices in mathematics.
Array Examples

- Houses on a Neighborhood Street
  - array size: 8
  - array index: house number
  - element type: house

Note: arrays don't need to be full (e.g., no house 0, 4, or 7)

- Sunlab Computers
  - array size: 72
  - array index: computer number
  - element type: computer

Note: Could be modeled as a 2-D array (see slide 47)

Outline

- Purpose
- Array Syntax
- ArrayLists
- Multi-Dimensional Arrays

Java's Syntax for Arrays (1/4)

```java
<type>[] <array-name> = new <type>[<size>];
```

declaration initialization

e.g., Dog[] dalmatians = new Dog[101];

- `<type>` denotes data type array holds: can be class, base type, interface, superclass, or another array (nested arrays)
- no reserved word "array"-[] brackets suffice
- We use new here, because arrays are a Java construct
- `<size>` must be integer value greater than 0; indices range from 0 to `<size>`-1
Java’s Syntax for Arrays (2/4)
- Arrays can be local variables, so they can get declared and initialized in single statement - just like objects and base types:
  ```java
  Colorable[] otherColorables = new Colorable[5];
  ```
- Arrays can also be instance variables, which get declared and then initialized separately in constructor:
  ```java
  private Colorable[] myColorables;
  ...
  this.myColorables = new Colorable[10];
  ```

Initializing an Array
- Houses on a neighborhood street
  ```java
  House[] houses = new House[8];
  ```
- Sunlab Computers
  ```java
  Computer[] sunlab = new Computer[72];
  ```
- Only array is initialized, not elements of array; all references are set to a default of null for Objects, 0 for ints, False for booleans, etc.

Java’s Syntax for Arrays (3/4)
- Accessing individual elements:
  ```java
  <array-name>[<index>]
  ```
  - index must be integer between 0 and (array size - 1)
  - result is value stored at that index
  - if `index` > size, or < 0, `ArrayIndexOutOfBoundsException` gets thrown
- Think of `student[i]` as the “name” of that particular student (like `student`) - simpler way to refer to each individual element in collection, better than having to use unique names

Note: some other languages allow an arbitrary value for the lower bound, but not Java!
Accessing Array Elements Example

- Houses on a Neighborhood Street
  ```java
  House[] houses = new House[8];
  House myHouse = houses[6];
  ```
- Sunlab Computers
  ```java
  CPU[] sunlab = new CPU[72];
  CPU myCPU = sunlab[42];
  ```

Java's Syntax for Arrays (4/4)

- An array element will work anywhere a variable would
  ```java
  myColorables[0] = new Ball();
  myColorables[2].setColor(Color.RED);
  Colorable myColorableVar = myColorables[3];
  this.myPaintShop.paintRandomColor(myColorables[4]);
  ```

Arrays as Parameters (1/3)

- Can pass entire array as parameter by adding array brackets to type inside signature
  ```java
  public int sum(int[] numbers){ //no size declared! }
  ```
- Now we can do the following (somewhere else in the class that contains sum):
  ```java
  int[] myNumbers = new int[5];
  System.out.println(this.sum(myNumbers));
  ```

Note: there is no way to tell from the use of `sum` that `myNumbers` is an array - you would need to see that `sum` and `Colorables` were declared to know that.
Arrays as Parameters (2/3)

- How do we determine size of array?
  - arrays have `length` as a public property (not a method)
  - use special "dot" syntax to determine `length`; here we inquire it, then store it for later

```java
int arrayLength = <array-name>.length;
```

Arrays as Parameters (3/3)

- How does `.length` work in actual code?

```java
public int sum (int[] numbers){
    int total = 0;
    for (int i=0; i < numbers.length; i++){
        total += numbers[i];
    }
    return total;
}
```

Note: for loop often used to traverse through all elements of array. Can use loop counter (`i` in this case) inside the body of loop but should never reset it. Incrementing/decrementing counter is done by for-loop itself!

Example: Watching Survivor Seasons (1/2)

- We want to watch all 41(!) seasons of Survivor one by one, using our array of `SurvivorSeasons`
Example: Watching Survivor Seasons (2/2)

```java
// first, declare and initialize the array
SurvivorSeason[] seasons = new SurvivorSeason[41];

// then, initialize the contents of the array
seasons[0] = new SurvivorSeason("Borneo");
seasons[1] = new SurvivorSeason("The Australian Outback");
...;
seasons[40] = new SurvivorSeason("Survivor 41");

// lastly, use a loop to play the seasons
for (int i = 0; i < seasons.length; i++) {
    seasons[i].play();
}
```

ArrayIndexOutOfBoundsException (1/2)

- Careful about bounds of loops that access arrays!
- Java throws `ArrayIndexOutOfBoundsException` if index is negative since sequence starts at 0
- Also throws `ArrayIndexOutOfBoundsException` if index is ≥ array size; remember that array goes from 0 to `n-1`

ArrayIndexOutOfBoundsException (2/2)

Example of a classic "off-by-one" error!

```java
// first declare and initialize the array
SurvivorSeason[] seasons = new SurvivorSeason[41];

// then, initialize the contents of the array
seasons[0] = new SurvivorSeason("Borneo");
seasons[1] = new SurvivorSeason("The Australian Outback");
...;
seasons[40] = new SurvivorSeason("Survivor 41");

// lastly, use a loop to play the seasons
for (int i = 0; i <= 41; i++) {  // should be i < 41
    seasons[i].play();
}
```

In Terminal:
```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:
    at Survivor.play(Survivor.java:64)
```

Note: The error tells you which index is throwing the error. Here, the loop is attempting to access the element at index=41, but our largest index of an array of size 41 is `n-1` or, in this case, 40.
TopHat Question
Consider the sum function from slide 19:
public int sum (int[] numbers){
    int total = 0;
    for (int i=0; i < numbers.length; i++) {
        total += numbers[i];
    }
    return total;
}
What if the code read i <= numbers.length?
A. It would wrap around and add the value at index 0 again
B. It would reach the last element of the array
C. It would raise an ArrayIndexOutOfBoundsException
D. None of the above

for vs. for-each loop (1/4)
● Intended to simplify most common form of iteration, when loop body gets applied to each member of collection
● How do for-each loop and for loops differ?
  o for loop gives access to index where item is stored
  o for-each loops don’t have direct access to index, but can easily access item (see next example)

for vs. for-each loop (2/4)
● for loops were extended to for-each (or for-in) loops, which iterate over the contents of a data structure rather than indices

```java
for (<type> <var>: <structure>){
    <loop body>
}
```

- `<type>`: class of objects stored in the `<structure`
- `<var>`: name of current element-holds each successive element in turn
- `<structure>`: data structure (array or other collection) to iterate through
**for vs. for-each loop (3/4)**
- If every element needs to be iterated and loop body doesn't need element index, for-each loops suffice:

```java
for (SurvivorSeason season : seasons){
    //notice we don't need to use index to get members from ArrayList
    season.play();
}
```

- Great advantage of for-each loops is that they don't raise ArrayIndexOutOfBoundsException! Why?
  - Java does the indexing for you!

**for vs. for-each loop (4/4)**
- Consider this for loop:

```java
for (int i=0; i < seasons.length; i++){
    if (i % 2 == 0) {
        //if index 'i' is even
        seasons[i].play();
    }
}
```

- Only want to watch seasons of survivor with even index so for-each loop wouldn't work.
  - we don't execute play() on every element in the array, we only care about elements at specific indices.

**Adding and Deleting in Arrays (1/2)**
- When adding at particular index, all other elements falling in and after that index must get shifted right by programmer (their indices are incremented by 1) otherwise data at index of insertion will be erased and replaced.
Adding and Deleting in Arrays (2/2)

- When deleting from particular index, all other elements falling in and after that index must get shifted left by programmer to fill the newly opened space (index decremented by 1)

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>23</td>
<td>32</td>
<td>67</td>
<td>56</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>
```

Outline
- Purpose
- Array Syntax
- ArrayLists
- Multi-Dimensional Arrays

java.util.ArrayList (1/2)
- java.util.ArrayLists, like arrays, hold references to many objects of same data type
- Another kind of collection, also using an index, but much easier management of making changes to array at runtime
- As name implies, it has properties of both arrays and Lists (covered later)
- Differences with arrays:
  - don’t need to be initialized with size - can hold an arbitrary and mutable number of references
  - are Java classes, not Java constructs, so have methods
java.util.ArrayList (2/2)

- Why use them instead of arrays?
  - when number of elements to be held is unknown
  - storing more data in an array that’s too small leads to errors
  - making array too large is inefficient, takes up more memory than necessary
  - handles update dynamics (shifting elements in memory) for you

- Why use arrays instead of array lists?
  - want something simple
  - want to use less memory (when you expect both array and array list to hold same number of elements)
  - want faster operations

Objects

- ArrayLists, like arrays, can hold any Object!

- Every class implicitly extends Object
  - every object is an Object

- Object is the most generic type possible
  - Object django = new Dog();
  - Object pongBall = new CS15Ball();
  - Object cartoonPane = new Pane();

What can ArrayLists hold?

- Upside: ArrayLists store things as Object—maximum polymorphic flexibility
  - since everything is an Object, ArrayLists can hold instances of any and every class: total heterogeneity
  - easy adding/removing anything

- Downside: ArrayLists only store Objects:
  - only methods available are trivial ones of Object itself: equals(), toString(), and finalize()
  - typically want homogeneous collection to store only objects of particular type (and its subtypes) AND have the compiler do type-checking for that type to enforce homogeneity
Generics! (1/2)

- Generics allow designer to write collection class A to hold instances of another class B, without regard for what class B will be (can be any Object for ArrayList). User of that class A then decides how to restrict/specialize type for that homogeneous collection.
- This is the constructor of the generic ArrayList (a collection class):
  ```java
  public ArrayList<ElementType>();
  ```
- Think of ElementType as a “type parameter” that is used as a placeholder that the user will substitute for with any non-primitive type (class, interface, array, …)
  - primitive types: boolean, int, double must be special-cased – Slide 42
- For example, we saw the use of generics to specialize implementation of EventHandler in interface to handle a specific type of Event, e.g.,
  ```java
  EventHandler<ActionEvent>
  ```
- Provides flexibility to have collection store any type while still having compiler help by doing type-checking.

Generics! (2/2)

- With generics, ArrayList was implemented by the Java team to hold any Object, but once an instance of an ArrayList is created by a programmer, they must specify the type. Let’s create an ArrayList of SurvivorContestants for our Survivor: Rhode Island theme!
  ```java
  ArrayList<SurvivorContestant> contestants = new ArrayList<>();
  ```
- We specify SurvivorContestants as the type that our ArrayList, contestants, can hold. Java will then replace ElementType with SurvivorContestant in ArrayList method parameters and return types.
- Can think of generics as a kind of parameter, just with different syntax (the <>), since only methods have parameters, not classes. In this case, ElementType acts as the formal parameter and SurvivorContestant is the argument.
- Generics, like classes and methods with parameters, provide generality in programming! (as does polymorphism in parameter passing).

java.util.ArrayList Methods (1/6)

```java
//Note: only most important methods shown (All defined for you!)
//see Javadocs for full class
//Note: literal use of < and >, only on the constructor; most methods use the specified ElementType

public ArrayList<ElementType>();
```

//one of the many constructors for ArrayList class - specialize
//it by providing ElementType, just as Array has the type it
//stores. Note: <> and > are literal - think of them as “of type”

```java
public ElementType get(int index)
```
java.util.ArrayList Methods (2/6)
//Two add methods with unique method signatures - example of method overloading
public boolean add(ElementType element)
    //inserts specified element at end of ArrayList

public void add(int index, ElementType element)
    //inserts the specified element at the specified position in this ArrayList; just as with arrays, causes indices of elements "to the right" to be incremented - but is done automagically */

public ElementType remove(int index)
    //removes the ElementType at given index and returns it

java.util.ArrayList Methods (3/6)
public int size()
    //returns number of elements stored in ArrayList

public boolean isEmpty()
    //returns true if ArrayList contains zero elements; false otherwise

java.util.ArrayList Methods (4/6)
• ArrayLists also have methods that access elements through search (as opposed to using an index)
  • these methods take parameter of type Object
  • but should never pass in anything besides ElementType
java.util.ArrayList Methods (5/6)

public int indexOf(ElementType elem)
//finds first occurrence of specified element, returns -1 if element not in ArrayList

public boolean contains(ElementType elem)
//return true if ArrayList contains specified element

public boolean remove(ElementType elem)
//remove first occurrence of specified element and returns true //if ArrayList contains specified element

java.util.ArrayList Methods (6/6)

- Some other ArrayList notes...
  - can add object in particular slot or append to end
  - can retrieve object stored at particular index and perform operations on it
  - can use for or for-each loop to access all objects in ArrayList
  - shifting elements for adding/deleting from ArrayList is done automatically by Java!
    - beware that indices past an insertion/deletion will increment/decrement respectively

ArrayList Example (1/2)
- Store an ArrayList of baking items in your pantry, using the Ingredient interface as the generic type
  ArrayList<Ingredient> pantry = new ArrayList<>();
  pantry.add(new Flour()); // inserts at back of list, index 0
  pantry.add(new Sugar()); // inserts at back of list, index 1
  pantry.add(1, new ChocolateChips()); // inserts at index 1
**ArrayList Example (2/2)**

```java
Ingredient mySugar = pantry.get(2); // returns Sugar instance
pantry.add(new BakingPowder()); // inserts at back of list, index 3
pantry.remove(mySugar); // removes Sugar instance
pantry.remove(0); // removes Flour instance
pantry.get(2); // raises ArrayIndexOutOfBoundsException
```

**Summary of ArrayLists (1/2)**

- More flexible than arrays for insertion/deletion
  - dynamically shifting elements and adjusting size in response to insert/delete is all done automatically
- Useful methods and return types:
  - `ElementType get(int index)`
  - `boolean add(ElementType element)`
  - `void add(int index, ElementType element)`
  - `int indexOf(ElementType elem)` // search
  - `ElementType remove(int index)`
  - `boolean remove(ElementType elem)`
  - `int size()`
  - `boolean isEmpty()`

- Weird edge case:
  - To make an ArrayList of primitive types, just specify Boolean, Integer, or Float in the generic brackets.
  - The `remove()` also has a weird edge case for Integers: you cannot use `remove(5)` to remove the 1st occurrence of 5, because it will treat it as the `remove()` method.
  - To remove an Integer element, use `remove(new Integer(<number>))`.

**Summary of ArrayLists (2/2)**

- Can hold heterogeneous collection of any kind of `Object`; want homogeneous collections...
- **Specialize** the ArrayList type by adding "generic" specification to a declaration or instantiation - thereby specifying two classes in one statement: the collection and the type of object it will hold and return

```java
ArrayList<SurvivorContestant> contestants = new ArrayList<>();
```

- Remember to use literal <> for specialized type!
TopHat Question
Which of the following uses an ArrayList correctly?

A. ArrayList<Contestant> contestants = new ArrayList<>();
   Contestant funnyContestant = new Contestant();
   contestants.add(funnyContestant);
B. ArrayList<ElementType> contestants = new ArrayList();
   Contestant toxicContestant = contestants[0];
C. ArrayList<ElementType> contestants = new ArrayList<>();
   Contestants fitContestant = contestants.first();
D. ArrayList<String> contestants = new ArrayList<>();
   Contestant villainContestant = new Contestant();
   contestants.add(villainContestant);

ConcurrentModificationExceptions

```java
public static void main(String[] args){
   ArrayList<SurvivorContestant> contestants = new ArrayList<>();
   contestants.add(new SurvivorContestant("Daniel"));
   contestants.add(new SurvivorContestant("Harriet"));
   contestants.add(new SurvivorContestant("Lila"));
   contestants.add(new SurvivorContestant("UV"));
   contestants.add(new SurvivorContestant("Will"));
   for (SurvivorContestant c : contestants){
      if(!c.getName().equals("Will")){
         contestants.remove(c);
      }
   }
}
```

- When trying to modify an ArrayList while iterating through it with a for-each loop, you will get a ConcurrentModificationException.
- Adding and removing cannot be done within a for-each loop because of the shifting of the elements in the list that Java does in response to an add or remove.
- Note: this is important for DoodleJump! We'll go over this issue in detail during the project help slides.

Outline
- Purpose
- Array Syntax
- ArrayLists
- Multi-Dimensional Arrays
Multi-Dimensional Arrays

- Modeling chess board:
  - not linear group of squares
  - more like grid of squares
- Multi-dimensional arrays are arrays of arrays of...
- Can declare array to be 2 (or more) dimensions, by adding more brackets
  - one pair per dimension
  - 2-D: int[][] grid = new int[a][b];
  - 3-D: int[][][] cube = new int[x][y][z];
  // a, b, x, y, z are ints whose values are set elsewhere

2-Dimensional Array Examples (1/2)

- Pixel Array
  - 2-D Array size: width by height
  - array indices: x, y
  - element type: RGB color
  - Pixel[][] MSFTlogo = new Pixel[x][y];

- Connect Four
  - 2-D Array size: 6 by 7
  - array indices: row, column
  - element type: checker
  - Checker[][] connect4 = new Checker[6][7];

2-Dimensional Array Examples (2/2)

- The Sunlab
  - 2-D Array size: 10 by 8 (approx.)
  - array indices: row, column
  - element type: computer
  - Computer[][] sunlab = new Computer[10][8];
Representing Multi-Dimensional arrays (1/2)

- Let's say we want to represent this grid of numbers:

```
1 2 3
4 5 6
7 8 9
```

Representing Multi-Dimensional arrays (2/2)

- How do we want to represent this grid? There are two equally valid options:

```
1 2 3
4 5 6
7 8 9
```
```
1 2 3
4 5 6
7 8 9
```

Ways to Think About Array Storage (1/3)

- Multi-dimensional arrays in Java do not make a distinction between rows or columns
  - think about 1D array – it doesn’t really matter if we call it a “row” or a “column”
  - can think of arrays as ordered sequences of data stored in contiguous positions in memory - no intrinsic geometry/layout implied
Ways to Think About Array Storage (2/3)

- Two visualizations of two-dimensional array (called ballArray) are equally valid. You can choose either for the organization of your array.

- Make sure there's consistency in the way you index into your 2-D array throughout your program!
  - since the elements are not stored in a specific order, the way that we insert elements and initialize and index into our array determines the order

<table>
<thead>
<tr>
<th>Row of Columns:</th>
<th>Column of Rows:</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Row of Columns" /></td>
<td><img src="image" alt="Column of Rows" /></td>
</tr>
</tbody>
</table>

Ways to Think About Array Storage (3/3)

- The choice between row-major and column-major organization can sometimes be arbitrary
  - Connect 4, a large carton of eggs, etc.
  - However, sometimes one will make more sense or simplify your program based on what you are trying to achieve
  - Can Storage example
    - goal: use array to keep track of the number of each type of can
    - makes most sense to use column-major organization
    - each column would be a sub-array of cans of the same type
    - rows within each column are either null (empty) or hold a can
    - can count number of each type by checking to see how many entries are full (or not null) in each sub-array (column, here)
  - For a table of entries use row major order, while for pixel (x, y) use column major order

TopHat Question

Here’s a grid of colored golf balls in column major order. What index is the light blue golf ball in?

A. ballArray[2][3]  
B. ballArray[2][1]  
C. ballArray[3][2]  
D. ballArray[1][2]
Example: Size of 2-D Arrays

```java
public static final int NUM_ROWS = 10; // defined in Constants
public static final int NUM_COLS = 6; // defined in Constants

public void practice2DArrays() {
    int numRows = myStringArray.length;
    int numCols = myStringArray[0].length;
    System.out.println("My array has " + numRows * numCols + " slots in total!");
}
```

Common Array Errors - Watch Out! (1/2)

- Cannot assign a scalar to an array
  ```java
  int[] myArray = 5; // 5 is not an array
  ```
  - to initialize array elements, must loop over the array and assign
    values at each index. Here we assign 5 to each element:
  ```java
  int[] myArray = new int[20]; // initializes array, not elements
  for (int i=0; i < myArray.length; i++){
    myArray[i] = 5;
  }
  ```
  Common Array Errors - Watch Out! (2/2)

- Cannot assign arrays of different dimensions to each other
  ```java
  int[][] myIntArray = new int[23];
  int[][] my2DIntArray = new int[2][34];
  myIntArray = my2DIntArray;
  ```
  - Doing so will result in this error:
    "Incompatible types: Can't convert int[] to int[]"
  - Similar message for assigning arrays of mismatched type
  - Take note that Java will automatically resize an array when
    assigning a smaller array to a larger one
2D Arrays Example (1/2)

• Let’s build a checkerboard with alternating black and white squares, using JavaFX
• Each square has a row and column index
• Let’s use row-major order
  o access any square with checkerboard[rowIndex][colIndex]
• JavaFX Rectangle’s location can be set using row and column indices, multiplied by square width factor
  o row indicates Y values, column indicates X value

2D Arrays Example (2/2)

```java
// instantiate a Pane and initialize the checkerboard 2D array
Pane myPane = new Pane();
Rectangle[][] checkerboard = new Rectangle[Constants.NUM_ROWS][Constants.NUM_COLS];
// loop through row and column indices
for (int row = 0; row < checkerboard.length; row++) {
    for (int col = 0; col < checkerboard[row].length; col++) {
        // instantiate rectangle, setting Y/X loc using row/col indices
        Rectangle rect = new Rectangle(col * Constants.SQ_WIDTH, row * Constants.SQ_WIDTH, Constants.SQ_WIDTH, Constants.SQ_WIDTH);
        // alternate black and white colors
        if ((row + col) % 2 == 0) {
            rect.setFill(Color.BLACK);
        } else {
            rect.setFill(Color.WHITE);
        }
        myPane.getChildren().add(rect); // graphically add the rectangle
        checkerboard[row][col] = rect; // logically add the rectangle
    }
}
```

SciLi Tetris: Loops and Arrays Writ Large

• In 2000, Tech House constructed then the largest Tetris game on the SciL – the Wiz flew out to play it!
• 5 months of work: 11 custom-built circuit boards, a 12-story data network, a Linux PC, a radio-frequency video game controller, and over 10,000 Christmas lights – see http://bastilleweb.techhouse.org/
• Video: https://www.youtube.com/watch?v=tkIRWoo9qrU&t=21s
• Article: http://news.bbc.co.uk/2/hi/science/nature/718009.stm
Announcements

- Cartoon deadlines
  - early handin: tonight, 10/21
  - on-time handin: Saturday, 10/23
  - late handin: Monday, 10/25
  - remember to tackle Minimum Functionality before trying any Bells & Whistles!

- DoodleJump partner form due tomorrow night
  - if you don’t fill it out, you’ll be assigned a random partner on no basis
  - if choosing your own partner, you must both fill it out with the correct logins; double-check there are no typos!

Topics in Socially-Responsible Computing
Autonomous Vehicles and Algorithmic Decisionmaking

Autonomous Vehicles

- Tesla “Full Self Driving” mode available to a small set of users to test on public roads (September 2021)
- National Highway Traffic Safety Administration began investigating Tesla in August about 12 crashes where autopilot drove into parked cars or emergency vehicles
- “In 2020, a Tesla with autopilot engaged experienced 0.2 accidents per million miles driven, while the US average was 9x higher.” - Tesla
Ethical concerns about “Full Self-Driving”

- Tesla has permit to operate autonomous vehicles with a human backup driver
- Tesla calls feature “Full Self-Driving”
- Consumer Reports: “Tesla will drive with no one in drivers’ seat”
- NHTSA: “ineffective monitoring of driver engagement” caused crash
- “NHTSA, which has shied away from imposing regulations for fear of stifling safety innovation, says that every state holds drivers accountable for the safe operation of their vehicles.” (AP)

Trolley problem

- Famous philosophical thought experiment about ethics
  - do nothing, trolley hits 5 people
  - pull switch, trolley only hits 1 person
  - variations on this
- Study at Michigan State: 90% of respondents would pull switch, 10% would not
- Survey of philosophers
  - 60.2% would pull switch
  - 7.4% would not pull switch
  - 24.2% other
- How do car manufacturers decide?
  - should cars protect their passengers at all costs? (even if killing pedestrians?)
  - need ethics committees!
- More on this in section next week!

Concerns about automation

- TuSimple, autonomous trucking company
  - estimates they will do driver-free demonstrations on public highways in 2021
  - UPS, Amazon, USPS ship freight with them
  - can get coast to coast in 2 vs. 4 days, only need to stop to refuel (not to mention safety!)
- 2.8 million workers drive trucks! Many jobs will likely disappear
- Small group that automates and huge group who lose jobs → huge inequality
Automation and responsibility

- Self-driving cars can save a ton of lives! But:
- What does it mean for a technological system to be "at fault"?
- Going into artificial intelligence:
- how does blaming "the tech"/"an algorithm" abstract away responsibility?
- we already make decisions informed by AI: i.e. cancer screening, driver's license scoring, etc. (more on this soon!)
- if the algorithm does a 'worse' job than a (competent) human, don't trust it!
- We should think about:
  - what political/economic/social contexts enable the development of certain technologies?
  - where (in its development, legitimization, etc.) were humans involved?
  - how (through regulation, labor organizing, ethics committees etc.) can we have agency over its development/not accept it as inevitable!!

More reading that may be of interest!

- "Automated trucking, a technical milestone that could disrupt hundreds of thousands of jobs, hits the road" — CBS / 60 Minutes (2021)
- "Map: The Most Common* Job In Every State" — NPR
- "Tesla's 'full self-driving' could be days away. Here's what you need to know." — Matt McFarland, CNN (2021)
- "U.S. regulator questions Tesla on the lack of a recall after an update to Autopilot." — Neal E. Boudette, NY Times (2021)
- "Elon Musk's problematic plan for 'full self-driving' Teslas" — Rebecca Heilweil, Vox (2021)
- "Self-driving cars have to be safer than regular cars. The question is how much." — Emily Stewart, Vox (2019)
- "Tesla is ordered to turn over Autopilot data to a federal safety agency." — Neal E. Boudette, New York Times (2021)
- "Wow, lot of interest in FSD beta..." — Elon Musk, Twitter
- "Opinion: No, 'truck driver' isn't the most common job in your state" — Rex Nutting, Marketwatch (2015)
- "CR Engineers Show a Tesla Will Drive With No One in the Driver's Seat" — Keith Barry, Consumer Reports (2021)
- "Regulatory Entrepreneurship" — SSRN (2017)
- "Tesla 'full self-driving' vehicles can't drive themselves" — Tom Krisher, AP (2020)
- "AI Ethicists Clash Over Real-World Aptness Of The Controversial Trolley Problem, But For Self-Driving Cars It Is The Real Deal" — Lance Eliot, Forbes (2020)
- "What Do Philosophers Believe?" — David Bourget and David Chalmers, 2013
- "Trolley Problem" — Wikipedia
- "Why we're still years away from having self-driving cars" — Eric Adams, Vox (2020)