GENETIC ALGORITHM IMPLEMENTATIONS
Paula and three of her friends are working on a lab to build a robot drone. Each person built a drone and named it (all different names), and one by one the drones were given a test flight. Unfortunately, each drone got stuck somewhere in the lab, including the one which landed on the projector. Determine the order in which each person launched his or her drone and where each got stuck.

1. Dave's drone, which isn't Buzz, didn't get stuck in the lights.

2. Cynthia's drone, which wasn't launched fourth, launched at some point after Zip.

3. Isaac's drone didn't get stuck in the instructor's desk.

4. Swoop was launched at some point after the drone that got stuck in the lights but at some point before Isaac's drone.

5. Buzz, which didn't get stuck in the lights, was launched at some point before the drone that got stuck in the projector screen (which wasn't Peekaboo).
There are four facts that together identify each of the four drones:
- Drone Name
- Owner Name
- Location
- Order Launched

Since we have four possible values for each of these, we have 16 total possible values.
Thus, we need an encoded string of length 16.
There will be four sections of the encoded string.
Each section will represent all four attributes for a specific drone.

MCHJ  NBGI  OAFL  PDEK
The position of each codon in each section corresponds to a specific type of fact in the problem.

- **MCHJ  NBGI  OAFL  PDEK**
- **Position 1: Order Codons (MNOP)**
- **Position 2: Owner Codons (ABCD)**
- **Position 3: Drone Codons (EFGH)**
- **Position 4: Location Codons (IJKL)**
Initial solutions are generated by randomly selecting codons that are not currently in the encoded string.

Example randomly generated encodings:
- MAHKNDGLOBFJPCEI
- MDHKNBFIOCGLPAEJ
- MDEJNCFLOAHIPBGK
- MBGKNAFIOCHJPDEL
After a solution is generated, it is sorted, ordering each section according to its order codon (from left to right).

Thus, the four sections will be ordered in the order of their order codons in the following way:

MXXX NXXX OXXX PXXX

This is done to simplify grading solutions.
OVERVIEW OF A GENERATION

- Grade each solution using the fitness function.
- Purge the bottom 10% of the population.
- Select the top 20% of the remaining population as elite.
- Randomly select 2 parents from elite and produce a child solution using crossover function. Parents from elite may only be used once per generation for crossover.
- Mutations may occur after crossovers.
- Place new child solutions into the population.
Each constraint is extracted from the drone problem. Create functions that enforce each constraint. Solutions have a starting fitness value of 0. Satisfying a constraint increases fitness value by 1.

Example constraint enforcements:

- “Dave’s drone isn’t Buzz.”
- Ensure that Dave and Buzz are not in same section.
- “Swoop was launched before Isaac’s drone.”
- Ensure that Swoop codon is before Isaac codon.
- Ensure that Swoop codon is not in same section as Isaac codon.
Randomly select one codon from either parent moving left to right.

If both codons have already been added to child’s encoding, randomly select an appropriate codon.
Randomly select a codon position (1 – 4).
Swap these codon values in two randomly selected sections.
AFTER GENERATIONS OF EVOLVING...

- # Generations: 163
- >> PERFECT ENCODING FOUND
- ====================
- FIRST, PAULA, ZIP, LIGHTS,
- SECOND, CYNTHIA, BUZZ, DESK,
- THIRD, DAVE, SWOOP, SCREEN,
- FOURTH, ISAAC, PEEKABOO, PROJECTOR,
A game simulation which utilizes genetic algorithms to evolve artificially intelligent agents that participate in the simulation.

The game being modeled in the simulation pits teams of players against one another in a fight to the death.

Players are able to inflict harm on their enemies and aid their allies through the use of spells and abilities, and whichever team suffers a casualty first loses.

Combat takes place in an enclosed space that players are able to move around in.

These are but just a few mechanics that add layers of complexity to the game.

The goal behind the simulation and its evolving players is to discover what actions by players result in repeated victory.
Within the game there exist players, whose sole purpose is to combat one another using various spells and abilities.

Three varying types of players exist in the game, each allowing the player to utilize different spells and abilities than the other player types.

A team consists of one player of each type, with all three teammates working together.

Two teams compete against each other in a match, for a total of six players competing in three versus three combat.

A match between two teams takes place on a map, an enclosed area that contains obstacles that obstruct a player’s movement and vision.

Players are capable of moving around the map freely, utilizing obstacles to hide from one another as necessary.

In order to win a match, one team must successfully defeat another player on the enemy team.
From the previous slide’s list of game mechanics we can identify components of the game that require implementation in the simulation.

- Players.
- Spells.
- Obstacles.
- Map.
- Matches.
- Teams.
PLAYERS

- Have several attributes:
  - Health (HP). Must remain above 0 or player’s team loses.
  - Mana (MP). Required to use spells and abilities.
  - Spell Priority Queue. Informs the player of spell precedence.
  - Hide Value. Informs the player what HP to hide at.
  - Heal Value. Informs the player what HP to heal at.

- Spells available to the player vary depending on their type.
- There exist three types of players.
  - One may heal anyone on the team and deal support damage.
  - The other two may deal heavy damage to opponents and self heal occasionally.
Have several attributes which directly modify the target player.
Spells are unique to each type of player.

Spell effects:
- Reduce HP of target.
- Increase HP of target.
- Reduce movement speed of target.
- Increase movement speed of target.
- Amplify effects of other spells on target.
- Dampen effects of other spells on target.

A player’s spell priority queue dictates which spells are used in which order.
OBSTACLES

- Four exist on the map and are stationary.
- Obstruct player’s movement and vision (line of sight).
- Used tactically to position oneself and hide from enemies.
MAP

- The vector space in which players and obstacles exist.
- Players move through the vector space.
- Obstacles are stationary.
Matches consist of two teams competing against one another.

A match ends when a team member on a team drops to zero or less.
Teams consist of three players each, one of each type.
Twenty teams are created randomly.

Two groups of ten are selected and placed together in a list.

When the initial teams are created, the players on the team have the following attributes randomly selected:

- Spell priority queue.
- Hide value.
- Heal value.

Each group of teams will remain together for the remainder of the simulation.

Both groups will therefore be competing against one another.
Create a list of matches, selecting teams from each group, paring them against one another round robin style.

Play each match, adding the results of the simulation to each team’s fitness value.

After every match is played, divide the fitness value for each team by the number of matches the team played (therefore getting the team’s average fitness).

Sort each group’s teams according to their average fitness, with higher fitness scores being first in the list.

Store the top team of each group, the “alphas”.

Prune the bottom 50% of each group, but store the top team from the list of pruned teams (the “runt”).

Perform crossover with the alpha of each group with the other members of each group including the runts.

Chance for mutations after each crossover.
CROSSOVER

- Select spells in order from both parents.
- Take the first spell from the alpha first, followed by the first spell from the second parent.
- Repeat for each spell in the priority list.
- Skip any spells that already exist in the priority list.

Example:

- A B C D E F
- B F E D A C
- A B F C E D
Select a random position in the priority queue.
Select randomly to swap left or right.
Swap the randomly selected position with the value to its left or right.

Example:
Random Position: 3
Random Direction: Right

A B F E C D
A B F E C D
The top team for each group will hopefully have discovered the best values for:

- Spell priority list.
- Hide HP.
- Heal HP.