# Project 3: Cross-Country Canada

Using variables, lists, functions, and conditionals in Python, students will celebrate Canada by creating their own unique variation of the classic Cross-Country Canada game.
(**https://en.wikipedia.org/wiki/Cross-Country_(video_game_series)#Crosscountry_Canada_(1986))**

## Overview

Cross-Country Canada was a text-based video game popular in the 90's. This beloved retro game was both educational and entertaining. At the start of the game, the player is given a goal to deliver (by truck) commodities from one Canadian city to another. To achieve your goal, the player must reference a city-commodity cross reference chart and also know which commands are recognized:
**https://gamefaqs.gamespot.com/pc/566644-cross-country-canada/faqs/30240**

In this project, we will create a simplified single-player version of the game.

## Details

### Behavior (Suggestions for the Basic Version)

- The player starts the trip in Vancouver, BC and ends the trip in Halifax, NS. The player will only travel in one direction.
- The player is given one commodity to deliver.
- The player must navigate the truck to different cities to pick up the commodity, and deliver it to the destination, within 30 days.
- At the beginning of the game, user is asked their name.
- Each turn, the player is asked what action they choose, where the player can type in one of the following commands:
  - `TRAVEL (T)`: moves you randomly between 500-1200 km and takes 1-3 days (random)
  - `REST (R)`: increases health 1 level (up to 5 maximum) and takes 1-3 days (random).
  - `BUY FOOD (B)`: buys food between 50-150 kg (random) and takes 1 day
  - `GET (G)`: pick up the commodity at the city.
  - `STATUS (S)`: lists location, health level, distance traveled, food available, commodity picked up (if any) and number of days travelled.
  - `HELP (?)`: lists all the commands.
  - `QUIT (Q)`: will end the game.
- Some possible assumptions:
  - Limit the map to have only 7-10 major cities.
  - Limit each city to have only 1 commodity
  - The player eats 5 kgs of food a day.
  - The player's health decreases by a random amount every few moves.

**Emphasize with students**

**BC ADST Computer Programming 11 Curriculum Competencies - Applied Technologies, Applied Design**

In this project, we are creating a software simulation of a real-world activity and within a real-world geographical context. As you design your game, be sure to research the location, roads, and natural resources associated with your chosen locations, to make the game as realistic as possible. One advantage of software is ease of accessibility by internet transfer. We can share digital creations/games with players who live even in remote or un-noticed locations. At the same time, all players are introduced to, and educated about, the valuable resources of our land and the incredible vastness of our nation.

Sharing your digital creation provides opportunity for collecting feedback, cross-pollination ideas, and direct future development iterations. Find a school in another part of the country who might be interested to play / test out some of your class projects. Provide a quick survey of questions to relating to their user experience, joys and frustrations, together with ideas for future improvements.

---

## Implementation details

- Document the behaviour of your game. This includes the list of commands, and list of city/commodities and assumptions
- First describe the "basic" version. Then, optionally, describe "future release" possibilities.
- Show this proposal to your teacher, to make sure that the scope is suitable.

---

**Emphasize with students the following:**

**BC ADST Computer Programming 11 Big Idea - Design Cycle**

Every project must have a scope. This is an initial document or plan of what your software is supposed to do, or will do. Before you begin the design and coding of this game, write down the behaviour of your game. You can use the suggestions above, or modify it to be more unique. However, keep your scope simple and clear. Resist the urge to overly complicate the game in your scope definition. Once you get a basic version that is working, you can add new commands or features in a future revision! Software development is iterative, and scaffolds over time.

---

## Grading

| **Functional Correctness (Behavior)** | |
|---|---|
| `TRAVEL`, `REST`, `BUY FOOD`, `GET` | 15 |
| `STATUS`, `HELP`, and `QUIT` | 5 |
| Game ends according to criteria defined in the documentation | 10 |
| Days roll over correctly | 10 |
| Food decreases accordingly | 5 |
| Health decreases randomly | 5 |
| **Sub total** | 50 |

| **Technical Correctness** | |
|---|---|
| Correctly use functions and contracts | 20 |
| Correctly use imported random function | 5 |
| Correctly use global variables | 5 |
| Correctly use and update variables | 5 |
| **Sub total** | 35 |

| **Design and Documentation** | |
|---|---|
| Clearly states game rules and assumptions | 5 |
| Provides good comments and documentation to the code | 10 |
| **Sub total** | 15 |

| | |
|---|---|
| **Total** | 100 |

## Extra Credit: Possible Advanced Features

1. Add a `fill up` command that includes gas consumption as part of the game.
2. Create events that occur randomly, like a forest fire roadblock, that will affect health and time.
3. Allow the user to travel East or West. Implement the `travel east (te)` and `travel west (tw)` commands.
4. Allow the user to choose the number of commodities to deliver, and make sure that the commodities assigned are all different.
5. Make the rate of food consumption be a function of the day of the week

## Extension

Instead of an East-West only map of Canadian cities, create a map that is a grid (eg. 2x5 grid of cities), or some other network graph. The game can provide hints or feedback on the routes chosen based on various factors including distance, and time.

## Emphasize with students some more

**BC Mathematics Computer Science 11: Algorithms, Computational Thinking, Solving Problems**

Pathfinding is the job of finding the shortest (based on a weight or cost associated with each path segment) path between two points on a network graph. The weight or cost would be based on some criteria (such as distance, time, scenery). Pathfinding algorithms are used in many applications: artificial intelligence, navigation, transportation, game design, virtual reality, etc. Ask students to discuss possible solutions and the reasoning behind them. Use graphs to visualize.

As our networks become bigger, the steps for finding a good path are computationally expensive. This is an ideal job for the computer!

---