AMS 562/DCS 522 Scientific Programming in C++
Team Projects

Joseph Schuchart

Fall 2025

Chapter 1

Introduction

The AMS 562 course, “Scientific Programming in C++”, employs team projects to enhance learning in programming, computation, and the application of software tools. These projects offer students opportunities to extend their skills.

Students are required to independently form teams, each composed of two to three members. It is crucial for all team members to participate actively and contribute to the learning process. In rare cases, students may be assigned or reassigned to teams by the instructor to ensure effective team dynamics and equitable learning opportunities for all participants.

Given the short timeframe allocated for these projects, students must be mindful of the semester’s time constraints when developing their project ideas. Important dates to remember include:

- **Project Mid-Term Report (Nov. 17th):** Each team will present their progress in person and receive feedback from the instructor and peers.
  - Slides are due by **midnight on Nov. 16th**.
  - The report is due by **midnight on Nov. 17th**.
- **Final Presentations (Dec. 3rd):** The last class of the semester is reserved for final in-person project presentations.
  - Slides are due by **midnight on Dec. 2nd**.
  - The report is due by **midnight on Dec. 8th**.

In each session, each team will have approximately 15 minutes to present their work. These dates are crucial milestones for the project, serving as opportunities for teams to showcase their progress and for instructors to provide valuable guidance.

Collaboration, Version Control, and Integrating Generative AI

The use of version control is a fundamental requirement for team projects in AMS 562, with GitHub designated as the preferred platform. To ensure effective collaboration and project management, each team must:

- Create a new private GitHub repository specifically for their project.

- Add all team members as collaborators to this repository.
- Include the AMS562 GitHub account as a collaborator for instructor and teaching assistant access.

These steps facilitate efficient project management, progress tracking, and code reviews. Students are also encouraged to explore GitHub's advanced features, such as:

- Forks for managing independent development streams.
- Pull requests for merging code changes and facilitating discussions.
- Issues for task and bug tracking.
- Regular code reviews for quality assurance.

These practices not only enhance the development process but also instill best practices in software development and collaborative teamwork.

In parallel with collaborative coding, integrating generative AI tools like GitHub Copilot and ChatGPT is encouraged. These tools should be seen as virtual assistants, supporting the team with tasks such as brainstorming, project scope refinement, hypothesis generation, debugging, code review, and documentation enhancement. It is crucial for teams to critically assess the outputs from these AI tools for accuracy and relevance. Ethical use and appropriate crediting of contributions from AI tools in project reports are mandatory to maintain academic integrity.

Potential Projects

Each team in the AMS 562 course has the opportunity to either select from a range of predefined projects or propose their own, especially suitable for those actively engaged in research. The predefined project options, showcasing the diverse applications of C++ programming and scientific computing, include:

- Modeling the Spread of Infectious Diseases
- Simulating Gerrymandering in Redistricting
- Scheduling Amazon Delivery Trucks
- Analysis of Climate Change Using Temperature Data
- Predictive Modeling of Bacterial Color Dynamics
- Graph Reconstruction
- Rotating Numbers
- Hardest Maze
- Lines Game
- Traffic Controller

- 3D Surface Reconstruction

More details about these projects are provided in later chapters. Each project presents a challenge and learning opportunity in fields such as epidemiology, political science, logistics, and climate science.

Alternatively, a team can propose their own projects, particularly suitable for students actively engaged in research. These self-proposed projects offer flexibility to align with the team's specific interests and academic pursuits.

When selecting or proposing a project, teams should carefully consider the complexity and scale of the work in relation to their collective background and experience. Projects should be challenging enough to facilitate significant learning and skill development but not so overwhelming as to be unmanageable within the course timeframe. It is important to strike a balance where the project's difficulty and scope are commensurate with the team members' current capabilities and potential for growth. Additionally, the amount of work and the project's ambitions should be proportional to the size of the team. This approach ensures that all team members are adequately challenged and contributes to a fair distribution of responsibilities, fostering a productive and rewarding project experience for everyone involved.

For all projects, whether predefined or self-proposed, students are permitted to use other people's implementations as references or integrate other libraries, especially those written in C++, C, or FORTRAN. If using existing implementations or integrating other libraries, it is mandatory to properly cite these sources and give credit. For instance, if a public GitHub repository is used as a reference, this should be clearly stated in the project documentation. Besides utilizing other libraries, teams may also integrate other programming languages (such as Python, MATLAB, or Java) for pre- and post-processing tasks. However, it is crucial that a substantial core component of each project involves C++ programming of their own. This includes critical aspects such as software or algorithm design, data structures, and algorithm implementation.

Grading Rubric

Success in these projects is measured by both the achievement of initial goals and the learning journey undertaken. This journey includes acquiring new skills, running significant portions of the code, practicing effective teamwork, and developing interim conclusions relevant to the project theme. The communication of outcomes may require using various forms of visual aids, such as graphs, pictures, or videos.

The final project significantly influences the overall grade, accounting for 60% of it. The project comprises a mid-term report (20%) and the final project (40%). For the mid-term checkpoint, which contributes 20% to the total course grade:

- 5% for the presentation.
- 5% for a written report, spanning 3–4 pages, excluding graphics and references.
- 10% for establishing a working environment, including a collaborative repository on GitHub and the successful execution of preliminary steps.

For the final project, contributing 40% to the total course grade:

- 5% from responsible participation in peer grading of presentations.

- 10% based on the final presentation.
- 15% for the working code, demonstrated live.
- 10% for a comprehensive final report, 6–8 pages in length, not including graphics and references.

Both reports should encompass the project objectives, techniques and tools used, observations, conclusions, and a paragraph detailing the contributions of each team member.

During the project presentations, a peer grading process will be conducted. Each team will evaluate the others based on specific criteria to ensure a comprehensive understanding and fair assessment of the work done. The questions for peer grading are as follows:

- What were the software skills of the team at the start of the course?
- How clearly did the presentation describe the project objectives, activities, and outcomes (20%)?
- How much did you learn from the presentation and project activities (20%) (e.g., mathematical techniques, software tools, science, etc.)?
- How would you assess the project's difficulty and the progress the team made against the initial skills of the team and the number of team members (60%)?
- Briefly comment on one thing you especially liked about the project and/or presentation.
- Briefly comment on one thing you would suggest to the team to improve either their presentation or project.
- Any additional comments, such as whether any team member appears to be lacking contribution to the project.

These questions aim to evaluate not only the final outcomes but also the learning process, the difficulty level of the project relative to the team's initial skill set, the effectiveness of the presentation, and the overall contribution of each team member.

Chapter 2

Modeling the Spread of Infectious Diseases

This project is derived from Chapter 5 of *Projects for Scientific Programming in C++ and Other Languages* by Victor Eijkhout. It involves the development of a simulation model to study the spread of infectious diseases in a population through an agent-based model. Each individual in the population will be represented explicitly, allowing for detailed tracking of disease progression over time.

Model Design

Each person in the simulation will be modeled as an agent that can be in one of four states:

- **Susceptible**: Healthy but can become infected.
- **Infected**: Carrying the disease and can infect others.
- **Recovered**: Previously infected but now immune.
- **Vaccinated**: Immune without being previously infected.

The disease will spread from infected agents to susceptible neighbors based on the contagiousness of the disease. The model will be initialized with one infected agent and the remaining population susceptible. The simulation will run day by day, updating each agent's state until no infected agents remain.

C++ Programming Aspects

The project will encompass the following aspects of C++ programming:

- **Object-Oriented Programming**: Designing classes and objects (e.g., Population, Person) for the simulation.

- **Data Structures**: Utilizing arrays or vectors to manage the population data.
- **Control Structures**: Implementing loops and conditional statements for simulation progression and state updates.
- **Random Number Generation**: Using randomization for aspects like infection spread and vaccination.
- **File I/O**: Reading parameters from and writing results to files.

For detailed explanations and examples of these programming concepts, students are encouraged to refer to the relevant sections in Eijkhout's book.

Implementation

Develop the following core components:

- **Population**: A class to represent the agent population, infecting agents, and updating states each day.
- **Person**: A class representing each agent, managing state and infection spread.
- **Contagion**: Methods for probabilistic infection spread based on disease contagiousness.
- **Vaccination**: A method for random population vaccination.
- **Mutation**: Modeling disease mutation leading to reinfection possibilities.

Start with a simple model and incrementally add complexity with contagion, vaccination, and mutation aspects.

Analysis

Post-development, perform computational experiments to analyze:

- Vaccination rate impacts on outbreak metrics.
- Mutation dynamics and immunity to different strains.
- Herd immunity effects.

Use varied model parameters, large populations, multiple replications, and statistical methods for robust analysis.

Team Collaboration and Task Partitioning

The project work may be divided among team members as follows:

- **Model Development**: Designing the disease spread algorithms and implementing the agent states.
- **Data Management**: Handling population data and statistics.
- **Simulation Testing**: Running the simulation and analyzing initial results.
- **Feature Implementation**: Adding advanced features like mutation and vaccination.
- **Data Analysis and Reporting**: Analyzing final results and preparing reports.

Conclusion

This project applies C++ programming to epidemiology, enhancing understanding of population dynamics in disease spread and containment strategies.

Chapter 3

Simulating Gerrymandering in Redistricting

This project is derived from Chapter 7 of *Projects for Scientific Programming in C++ and Other Languages* by Victor Eijkhout. It involves building a computational model to explore gerrymandering in political redistricting, where district boundaries are manipulated for political advantage.

For background reading, see `https://redistrictingonline.org`.

Model Design

Develop an abstract model representing voters, districts, and redistricting strategies. Key components include:

- **Voters**: Agents with party affiliations.
- **Districts**: Groupings of voters into contiguous districts.
- **Population**: Collection of all voters in a state.
- **Redistricting**: Process of redrawing district boundaries.

Initialize a population of voters with party affiliations and implement algorithms for partitioning them into districts.

C++ Programming Aspects

The project will cover:

- **Object-Oriented Programming**: Creating Voter and District classes.
- **Data Structures**: Using arrays or vectors for managing voters.
- **Algorithm Design**: Implementing districting algorithms.

- **File I/O**: Handling input data and output results.

For detailed programming concepts, students should refer to Eijkhout's book.

Implementation

Incrementally build the model components:

- **Voter and District Classes**: Represent individuals and groupings.
- **Population Class**: Manage voter data and partisan lean.
- **Districting Algorithms**: Implement strategies for redistricting.

Begin with a 1D model, expanding to 2D and more complex scenarios.

Team Collaboration and Task Partitioning

Divide project work among team members:

- **Model Development**: Constructing the voter and district models.
- **Algorithm Implementation**: Developing districting strategies.
- **Simulation Testing and Validation**: Ensuring model accuracy.
- **Data Analysis**: Interpreting results and drawing conclusions.

Analysis

Perform experiments to quantify gerrymandering effects:

- Assess minority advantage in districting.
- Examine voting margins for partisan gain.
- Analyze the impact of constraints on district geometry and population.

Simulate redistricting with different populations and affiliations to understand gerrymandering dynamics.

Conclusion

Apply C++ programming to study gerrymandering for insights into this politically relevant issue.

Chapter 4

Scheduling Amazon Delivery Trucks

E-commerce companies like Amazon face complex logistical challenges in delivering customer orders efficiently. One such challenge is planning the routes for their delivery trucks to optimize factors like total distance traveled, number of trucks needed, and ability to meet promised delivery windows. In this project, we will build a simplified model of the Amazon delivery truck scheduling problem. The goal is to develop routing algorithms that minimize the total distance traveled by the trucks while respecting constraints like customers who have paid for next-day "Amazon Prime" delivery. This project is inspired by the exercises in Chapter 8 of *Projects for Scientific Programming in C++ and Other Languages* by Victor Eijkhout.

Problem Definition

We make the following simplifying assumptions:

- The delivery area is a 2D plane. Customer locations are specified by x, y coordinates.
- All trucks start and end at a central depot at coordinates (0,0).
- Each customer has a deadline specifying the last day their delivery can take place.
- "Amazon Prime" customers must receive their delivery the next day.
- For non-Prime customers, deliveries can be spread over multiple days.
- Deliveries cannot be split—each customer receives their full delivery in one truck visit.

With these assumptions, the problem is to schedule a given set of customer deliveries over multiple days and trucks to minimize the total distance traveled.

C++ Programming Aspects

This project will involve:

- **Object-Oriented Design**: Using C++ classes to represent delivery requests, truck routes, and delivery schedules.
- **Algorithm Implementation**: Coding algorithms for route optimization and delivery scheduling.
- **Data Structures**: Using appropriate C++ data structures for managing delivery data and route information.
- **Performance Optimization**: Techniques for efficient data processing and algorithm optimization in C++.
- **File I/O**: Handling input/output operations for loading delivery data and saving scheduling results.

Methods

We will implement the following components:

- **Address** class representing a customer delivery location.
- **DeliveryRequest** class representing a customer order with location and deadline.
- **Route** class representing a truck route as a sequence of addresses.
- **Schedule** class representing a multi-day plan of truck routes.

Key algorithms will include:

- Constructing single-truck delivery routes with a greedy nearest-neighbor heuristic.
- Improving routes by reversing route segments (Lin-Kernighan heuristic).
- Scheduling Prime vs. non-Prime deliveries optimally over multiple days and trucks.

We will test the algorithms on randomized delivery data. The main evaluation metric will be the total route distance for fulfilling a given set of deliveries under Prime vs. non-Prime constraints.

Team Collaboration and Task Partitioning

Tasks may be divided into:

- **Algorithm Development**: Building and optimizing routing algorithms.
- **Class Implementation**: Coding the primary classes in C++.
- **Data Management**: Handling input data and results.
- **Testing and Validation**: Running simulations and validating results.
- **Documentation and Reporting**: Preparing comprehensive documentation and reports.

Applications

This project will provide hands-on experience with common vehicle routing problems and algorithms. The code can serve as a starting point for more realistic scheduling models that consider factors like truck capacity, delivery time windows, traffic, and driver breaks.

With additional data on costs and delivery priorities, the model could be extended to optimize objectives beyond distance, such as minimizing fuel costs or maximizing on-time deliveries.

Conclusion

Vehicle routing is a rich problem domain that combines optimization, algorithms, and logistical constraints. This project will explore core techniques for efficient delivery scheduling in a simplified model relevant to e-commerce companies. The methods can be expanded in many directions to tackle real-world transportation challenges.

Chapter 5

Analysis of Climate Change Using Temperature Data

This project is derived from Chapter 11 of *Projects for Scientific Programming in C++ and Other Languages* by Victor Eijkhout. There is an ongoing debate around whether the Earth's climate is changing significantly over time. In this project, we will analyze historical temperature data to look for statistical evidence of climate change trends. The core analysis will involve testing the hypothesis that temperature extremes (highs and lows) follow a stationary statistical distribution over time against the alternative hypothesis of an increasing linear trend over time.

Data

The analysis will use the NASA GISS Surface Temperature (GISTEMP) dataset, containing global average monthly temperature anomalies back to 1880.

The temperature anomalies represent deviations from a baseline average temperature computed over the 1951-1980 period. Positive anomalies indicate warmer than average temperatures.

Methods

The following statistical analyses will be applied to the temperature data:

- Test whether the gaps between record high temperatures are increasing over time, as would be expected with non-stationary warming.
- Fit a linear trend to the gaps between high temperature records over time and assess the model fit.
- Apply formal hypothesis testing to determine if the linear trend model is statistically significant compared to a stationary null hypothesis.

The analysis will leverage C++'s array processing capabilities applied across the data for each month and year.

C++ Programming Aspects

This section will explore:

- **Data Handling**: Using C++ to efficiently process and manage large temperature datasets.
- **Statistical Analysis**: Implementing statistical methods in C++ for hypothesis testing and trend analysis.
- **Visualization**: Developing C++ code for data visualization to illustrate temperature trends.
- **Performance Optimization**: Techniques for optimizing data processing and analysis routines in C++.
- **Interoperability**: Integrating C++ with other tools and languages for advanced data analysis.

Team Collaboration and Task Partitioning

Tasks may include:

- **Data Analysis**: Conducting the core statistical analyses.
- **C++ Development**: Implementing analysis routines and data processing in C++.
- **Visualization and Reporting**: Creating visual representations of data trends and documenting findings.
- **Optimization**: Focusing on performance enhancements and testing.
- **Project Management**: Overseeing project progress and coordinating team activities.

Conclusions

This project provides an opportunity to apply statistical techniques for hypothesis testing and trend analysis to a real climate science problem. The results will shed light on the debate around recent climate change patterns using historical data.

The general methodology could be extended to other climate variables like precipitation, sea level rise, or storm severity. The C++ programming approach is well-suited to additional statistical analyses as well.

Chapter 6

Predictive Modeling of Bacterial Color Dynamics

This project involves developing a C++ program to simulate and predict the color evolution of a population of bacteria placed on a grid. Each bacterium changes its color based on the colors of its neighbors within a specific distance. The challenge is to accurately predict the original colors of the bacteria after a given number of days. More details of this project can be found at TopCoder Marathon Match 112.

Objectives

1. **Algorithm Development**: Design and implement algorithms that accurately predict the initial state of the grid based on its state after a given number of days.
2. **Data Processing**: Efficiently process grid data to determine the color of each bacterium.
3. **Scientific Analysis**: Apply principles of scientific programming to analyze the color dynamics of the bacteria.
4. **C++ Proficiency**: Utilize advanced features of C++ for efficient computation and data handling.

Project Specifications

- **Input Parameters**:
  - N: Size of the grid ($N \times N$).
  - C: Number of different bacteria colors.
  - D: Number of days for the simulation.
  - K: Distance threshold for influencing colors.

- **Data Representation**: The grid is represented as an $N \times N$ matrix, with each cell containing a value corresponding to the color of the bacterium.
- **Color Change Rule**: A bacterium changes its color based on the minority color of its neighbors within a Euclidean distance of K.
- **Output**: Predicted initial color grid before the simulation started.

Methodology

- **Algorithm Design**: Develop algorithms to simulate color change and reverse-engineer the initial state.
- **Data Handling**: Implement efficient data structures for grid management.
- **Optimization**: Use C++ features like pointers, dynamic memory allocation, and standard template libraries for efficient coding.
- **Testing**: Create various test scenarios to ensure the algorithm's correctness and efficiency.

C++ Programming Aspects

- **Advanced C++ Features**: Utilizing pointers, dynamic memory allocation, and standard template libraries.
- **Data Structures**: Implementing efficient structures for managing grid data.
- **Algorithm Implementation**: Developing and coding algorithms for color prediction and simulation.
- **Performance Optimization**: Enhancing code for speed and memory efficiency.
- **Testing and Debugging**: Creating test scenarios to ensure algorithm accuracy and efficiency.

Team Roles

1. **Algorithm Strategist**: Focus on developing the core algorithm for color prediction.
2. **Data Manager**: Handle data input/output and manage grid data structures.
3. **Code Optimizer**: Optimize the code for performance, focusing on memory and speed.
4. **Testing and Validation Specialist**: Develop test cases, validate the algorithm's accuracy, and debug.

Conclusion

This project not only aims to develop technical skills in C++ and algorithmic programming but also fosters teamwork and problem-solving in a scientific context. It provides a practical application of scientific programming principles, emphasizing data analysis, predictive modeling, and computational efficiency.

Chapter 7

Graph Reconstruction

This project involves the reconstruction of an undirected graph's structure using given path information. The task is to accurately predict the adjacency matrix of the graph based on the shortest path distances between pairs of nodes. The project will challenge students' ability to apply graph theory and algorithmic problem-solving in C++. More details can be found at TopCoder Marathon Match 115.

Objectives

1. **Understanding and Implementing Graph Theory**: Utilize graph theoretical concepts to model and solve the problem.
2. **Algorithm Development**: Develop algorithms to accurately reconstruct the graph from the provided path information.
3. **Optimization Techniques**: Optimize the algorithm for the best F1 score, balancing precision and recall.
4. **C++ Programming**: Apply C++ programming skills to implement efficient data structures and algorithms.

Project Specifications

- **Input**: Information about the shortest path distances between pairs of nodes in the graph.
- **Task**: Predict the adjacency matrix of the graph, representing connectivity between nodes.
- **Scoring**: Use the F1 score to evaluate the quality of the graph reconstruction.

Methodology

- **Graph Modeling**: Model the problem using appropriate graph structures.

- **Algorithmic Approach**: Develop algorithms to infer graph connectivity from path information.
- **Optimization**: Implement strategies to maximize the F1 score.
- **C++ Implementation**: Efficiently implement the solution in C++.

C++ Programming Aspects

- **Data Structures**: Implementing efficient structures in C++ for graph representation.
- **Algorithm Development**: Coding algorithms for inferring the graph structure.
- **Performance Optimization**: Enhancing code for computational efficiency.
- **Testing and Debugging**: Creating test scenarios to ensure algorithm accuracy and efficiency.

Team Roles

1. **Graph Theorist**: Focus on understanding and applying graph theoretical concepts.
2. **Algorithm Developer**: Develop and implement the core algorithm for graph reconstruction.
3. **Optimization Expert**: Optimize the algorithm to improve the F1 score.
4. **C++ Programmer**: Implement the algorithm and data structures efficiently in C++.

Conclusion

This project provides an opportunity to apply graph theory and algorithmic skills in a practical scenario, enhancing students' problem-solving abilities and proficiency in C++ programming.

Chapter 8

Rotating Numbers

This project involves developing an algorithm to move numbers to their target locations on an $N \times N$ grid by rotating square subgrids. The goal is to minimize the total penalty incurred by the rotations and misplaced numbers, which involves strategic planning and optimization. More details can be found at TopCoder Marathon Match 117.

Objectives

1. **Develop an Algorithm**: Efficiently rotate subgrids and move numbers to their target locations.
2. **Apply Mathematical Concepts**: Calculate and minimize penalties associated with each move.
3. **Optimize the Solution**: Achieve the lowest total penalty.
4. **Implement Efficiently**: Use C++ for efficient computation and data handling.

Project Specifications

- **Task**: Rotate subgrids to move numbers to their target locations on a grid.
- **Penalty**: Calculate penalties for rotating subgrids and misplaced numbers.
- **Optimization**: Focus on minimizing the total penalty for the moves.
- **Implementation**: Develop the solution in C++, utilizing efficient data structures and algorithms.

Methodology

- **Algorithm Development**: Create algorithms to determine the optimal rotations and movements.

- **Mathematical Modeling**: Develop a model to calculate penalties and evaluate moves.
- **Optimization Techniques**: Implement strategies to minimize the total penalty.
- **C++ Implementation**: Code the solution in C++, focusing on efficiency and performance.

C++ Programming Aspects

- **Efficient Data Structures**: Utilizing appropriate C++ structures for grid management.
- **Algorithm Implementation**: Developing and coding algorithms for grid manipulation.
- **Performance Optimization**: Enhancing code for computational efficiency.
- **Testing and Debugging**: Ensuring algorithm accuracy and efficiency through rigorous testing.

Team Roles

1. **Algorithm Designer**: Focus on developing the core algorithm for rotating subgrids and moving numbers.
2. **Mathematical Modeler**: Handle the mathematical aspects of penalty calculation and minimization.
3. **Optimization Specialist**: Optimize the algorithm to minimize the total penalty.
4. **C++ Developer**: Implement the algorithm and optimize it for performance in C++.

Conclusion

This project provides hands-on experience in algorithmic problem-solving, mathematical modeling, and optimization in the context of a real-world challenge. It aims to enhance students' proficiency in C++ programming and their ability to tackle complex computational problems.

Chapter 9

Hardest Maze

Project Overview

The goal of this project is to design a maze on an $N \times N$ grid floor to challenge robot vacuum cleaners. The task involves strategically placing walls to maximize the total distance traveled by the robots, each having to visit multiple target locations. More details can be found at TopCoder Marathon Match 119.

Objectives

1. **Design a Challenging Maze**: Maximize the total distance traveled by robots.
2. **Implement Pathfinding Algorithms**: Determine the shortest paths for robots.
3. **Optimize the Maze Layout**: Increase the complexity of robot navigation.
4. **Develop Efficiently**: Use C++ for efficient computation.

Project Specifications

- **Task**: Create a challenging maze on an $N \times N$ grid.
- **Robots**: Design paths for multiple robots with specific target locations.
- **Optimization**: Aim to maximize the total distance traveled by the robots.
- **Implementation**: Develop and test the solution in C++.

Methodology

- **Maze Design**: Use computational techniques to design a challenging maze layout.
- **Pathfinding Algorithms**: Implement algorithms like A* or Dijkstra for robot navigation.

- **Analytical Optimization**: Use analytical methods to evaluate and improve the maze design.
- **C++ Development**: Implement and optimize the solution using C++.

C++ Programming Aspects

- **Efficient Data Structures**: Utilizing appropriate C++ structures for grid management.
- **Algorithm Implementation**: Developing and coding algorithms for grid manipulation.
- **Performance Optimization**: Enhancing code for computational efficiency.
- **Testing and Debugging**: Ensuring algorithm accuracy and efficiency through rigorous testing.

Team Roles

1. **Maze Designer**: Focus on the strategic placement of walls to maximize the maze's difficulty.
2. **Pathfinding Expert**: Implement algorithms to determine the shortest paths for robots.
3. **Optimization Analyst**: Optimize the maze design for maximum complexity.
4. **C++ Programmer**: Code the solution in C++, ensuring efficient execution and data handling.

Conclusion

This project provides a unique opportunity to combine creativity in maze design with algorithmic problem-solving and optimization in a scientific programming context. It challenges students to apply their C++ skills and computational thinking to develop an innovative and complex solution.

Chapter 10

Lines Game

In this project, students are challenged with a strategic and computational task focused on the "Lines" game, played on an $N \times N$ grid. This grid-based puzzle game requires players to arrange colored balls to form lines and maximize their score within a set number of moves. The challenge emphasizes algorithmic problem-solving and strategic thinking. For detailed information, visit the challenge page on Topcoder: Topcoder Marathon Match 125.

Game Mechanics

- **Grid Size and Colors**: The game utilizes a grid of size N and C different ball colors.
- **Line Formation**: Lines are formed with 5 or more adjacent balls of the same color.
- **Movement of Balls**: Players can move a ball to any empty cell connected by a path of empty cells.
- **Scoring**: Points are awarded based on the number of balls removed per move, calculated as $n^2 - 7n + 20$.
- **Game End**: The game concludes after 1000 moves or when no more moves are possible.

Objectives

- **Maximize Points**: The primary objective is to score as high as possible.
- **Strategic Movement**: Develop algorithms for optimal ball movement and line creation.
- **Algorithmic Efficiency**: Implement effective strategies for dynamic grid management.

C++ Programming Aspects

- **Algorithm Development**: Crafting strategies in C++ for ball movement and scoring.

- **Data Structures**: Utilizing appropriate C++ structures for grid and game state management.
- **Performance Optimization**: Enhancing code for computational efficiency in game strategy execution.
- **Testing and Debugging**: Ensuring accuracy and efficiency of algorithms and game mechanics.

Team Roles

- **Game Strategist**: Focus on developing effective gameplay strategies.
- **Algorithm Developer**: Create and implement algorithms for ball movement and line creation.
- **Optimization Expert**: Work on improving the efficiency of game strategies.
- **C++ Programmer**: Code the game mechanics and algorithms in C++ efficiently.

Conclusion

The Lines Game offers a unique blend of strategic planning, algorithmic problem-solving, and C++ programming, challenging students to apply their skills in a dynamic and engaging context. It provides a practical platform for enhancing computational thinking and optimization techniques.

Chapter 11

Traffic Controller

In this project, students will develop a program to manage traffic flow efficiently in a simulated city environment. The objective is to maximize the flow of traffic and minimize delays by controlling traffic lights at various intersections. More details about the project can be found at the Topcoder Challenge Link.

Project Goals

Understanding Traffic Flow Dynamics

Students will explore the dynamics of traffic flow in an urban grid. They will learn how traffic light control can impact the efficiency of traffic movement.

Algorithm Development

Develop algorithms to manage traffic lights in response to changing traffic conditions, aiming to optimize flow and reduce congestion.

Simulation and Modeling

Use simulation techniques to model traffic patterns and the impact of different traffic light control strategies.

Technical Details

The city is modeled as an $N \times N$ grid with R straight roads. Intersections are controlled by traffic lights, which can be switched between horizontal and vertical states. The program will run for a fixed number of turns, during which students must maximize the flow of traffic. Cars will appear on the roads and move towards their destinations, adhering to the state of traffic lights.

C++ Programming Aspects

- **Algorithm Implementation**: Coding traffic management algorithms in C++.
- **Data Structures**: Utilizing efficient C++ structures for modeling roads and intersections.
- **Simulation Techniques**: Implementing traffic flow simulation in C++.
- **Performance Optimization**: Enhancing code for computational efficiency in managing traffic.

Team Roles

- **Traffic Analyst**: Focus on understanding and analyzing traffic flow dynamics.
- **Algorithm Developer**: Create and implement algorithms for traffic light control.
- **Simulation Expert**: Develop and manage traffic simulation models.
- **C++ Programmer**: Code the solution in C++, ensuring efficient execution.

Conclusion

This project combines elements of urban planning, algorithmic problem-solving, and C++ programming, offering students a practical application of these skills in a simulated environment. It challenges students to apply their knowledge in a dynamic, real-world scenario, enhancing their abilities in computational thinking and strategy development.

Chapter 12

3D Surface Reconstruction

This project focuses on the challenge of reconstructing a hidden 3D surface using a limited number of samples. The detailed description of the project can be found at Topcoder's Marathon Match 147.

Objective

The primary objective is to reconstruct a 3-dimensional surface, which is created by the accumulation of multiple equations. Participants are required to minimize the number of samples while trying to reconstruct the surface as accurately as possible.

Technical Details

Surface Construction

- The surface is constructed using a set of randomly selected equations with randomly selected parameters.
- After the application of these equations, the surface is normalized so that each cell value falls within the range [0, 255].

Sampling

- Participants can perform a maximum number of samples calculated as floor($N \times N/9$).
- Each sample provides the values of a 3×3 subgrid centered at the sampling location.

Prediction

- After completing the sampling phase, participants need to predict the value of each cell on the surface.

- The prediction's accuracy is evaluated based on the difference between the predicted and actual surface values.

Scoring

The score depends on the normalized mean squared error between the predicted and actual surfaces and the number of samples used. The scoring formula is as follows:

$$\text{Score} = A \cdot \text{MSE} + (1.0 - A) \cdot \text{Sample Fraction},$$

where A is a constant, MSE is the mean squared error, and Sample Fraction is the fraction of samples used.

C++ Programming Aspects

- **Algorithm Development**: Designing algorithms in C++ to predict the 3D surface accurately.
- **Data Structures**: Employing efficient C++ structures for handling surface data and samples.
- **Performance Optimization**: Enhancing code for computational efficiency in data processing and prediction.
- **Testing and Debugging**: Ensuring the accuracy and efficiency of the prediction algorithms.

Team Roles

- **Mathematical Modeler**: Focus on developing the mathematical framework for surface construction and prediction.
- **Sampling Strategist**: Optimize the sampling process to maximize information gain.
- **Algorithm Developer**: Create and implement prediction algorithms.
- **C++ Programmer**: Efficiently code the solution in C++.

Conclusion

This project presents a complex and intriguing problem that combines elements of mathematics, algorithm design, and scientific computing. It offers a unique opportunity to delve into the world of 3D surface reconstruction and optimization.