

Programming Assignment 7

Attack of the Zurts

The Mechanoids of Planet Zork have captured you! To satisfy their lust for gruesome spectacles, they have thrown you in an arena with hungry zurts, a carnivorous Zorkian species. Well, that's the scenario for a new game under development. Your assignment is to complete the prototype that uses character-based graphics.

The three breeds of zurts have different color fur: red, yellow, and blue. When a zurt notices something of its color flying through the air in a particular direction, it alerts the others zurts of the same color, and those zurts move one in the direction that flying color is moving. Zurts have short attention spans, so if the flying color is no longer flying, they forget about moving in that direction.

The player fortunately has an unlimited collection of red, yellow, and blue fruit. A player wanting to try to get all the red zurts to move north, for example, will throw a red fruit north. Half the time, though, no red zurt would notice it, so nothing special happens, but the other half of the time, it's seen and all the red zurts move one step north, after which time the fruit has landed and the red zurts forget about moving north.

If you execute [this Windows program](#) or [this Mac program](#) or [this Linux program](#), you will see the player (indicated by @) in a rectangular arena filled with hungry zurts (each indicated by a R, Y, or B, its color). At each turn, you'll select an action for the player to take: move one step or stay put. The player will take the action. You will then select a fruit color and direction for the player to throw the fruit. Then each zurt will move. Normally, each zurt will move one step in a random direction. However, if the fruit is noticed by zurts of that color (half the time it won't be), those zurts of that color will move one step in the direction the fruit is moving. If a zurt moves onto the grid point occupied by the player, the player dies. (If the player moves to a grid point occupied by a zurt, the player dies, so that would be a dumb move.)

The grid points may have wall segments. If the player or a zurt moving of its own free will chooses a direction such that moving would be to a grid point occupied by a wall or to a point off the grid, then the player or zurt just doesn't move that turn. If a zurt moving because it's moving in the direction of the flying fruit attempts to move into a wall or off the grid, it doesn't move that turn, but suffers 1 unit of damage. A zurt that has suffered 3 units of damage is dead and is removed from the game. Clearly, it's in the player's interest to throw fruit in a way that causes zurts to smash into walls or the edges of the grid.

This smaller [Windows version](#) or [Mac version](#) or [Linux version](#) of the game may help you see the operation of the game more clearly.

At each turn the player may take one of these actions:

1. Stand. In this case, the player does not move.

2. Move one step north, east, south, or west. If the player attempts to move onto a wall or out of the arena (e.g., south, when on the bottom row), the result is the same as standing. If player moves to a grid point currently occupied by a zurt, the player dies.

The game allows the user to select the player's action: n/e/s/w for movement, or x for standing. The user may also just hit enter to have the computer select the player's move.

After the player moves, the user is prompted for a color and direction. You enter a two-character string such as "rn". The first character is the color (upper or lower case) and the second is n/e/s/w. The player throws the fruit, which flies only for the duration of the zurts' turn. (The flying fruit is not displayed on the screen.) At the zurts' turn, there is a 1/2 probability that the fruit will be noticed by a zurt of the same color. If it is, all zurts of that color move one step in the direction the fruit was thrown if they can; if they can't (because they'd hit a wall or move off the grid), they stay put, but suffer 1 unit of damage. If the fruit is not noticed by any zurt of the same color, the zurts of that color act like all the other zurts on that turn: They move normally.

Normally, each zurt picks a random direction (north, east, south, west) with equal probability. The zurt moves one step in that direction if it can; if the zurt attempts to move into a wall or off the grid, however, it does not move and suffers no damage. More than one zurt may occupy the same grid point; in that case, the display will show a color character indicating the color of one of the zurts at that point (the choice is arbitrary).

Your assignment is to complete [this C++ program skeleton](#) to produce a program that implements the described behavior. (We've indicated where you have work to do by comments containing the text TODO.) The program skeleton you are to flesh out defines four classes that represent the four kinds of objects this program works with: Game, Arena, Zurt, and Player. Details of the interface to these classes are in the program skeleton, but here are the essential responsibilities of each class:

Game

- To create a Game, you specify a number of rows and columns and the number of zurts to start with. The Game object creates an appropriately sized Arena and populates it with some walls, the Player, and the Zurts.
- A game may be played.

Arena

- When an Arena object of a particular size is created, it has no positions occupied by wall segments, zurts, or the player. In the Arena coordinate system, row 1, column 1 is the upper-left-most position. If an Arena had 10 rows and 20 columns, its lower-right corner would be row 10, column 20.
- You may tell an Arena object to put a wall segment at a particular position.
- You may ask an Arena object whether there's a wall segment at a particular position.
- You may tell an Arena object to create a Zurt at a particular position, of a particular color.
- You may tell an Arena object to create a Player at a particular position.
- You may tell an Arena object to have all the zurts in it make their move.
- You may ask an Arena object its size, how many zurts are at a particular position, and how many zurts altogether are in the Arena.

- You may ask an Arena object for access to its player.
- An Arena object may be displayed on the screen, showing the locations of the walls, zurts, and the player, along with other status information.

Player

- A Player is created at some position.
- A Player may be told to stand or to move in a direction.
- A Player may be told it has died.
- You may ask a Player for its position and its alive/dead status.

Zurt

- A Zurt is created at some position with some color.
- You may tell a Zurt to move.
- You may tell a Zurt to move in a particular direction. This is a forced move, and the zurt will suffer 1 unit of damage if it can't move.
- You may ask a Zurt object for its position, color, and alive/dead status.

The skeleton program you are to complete has all of the class definitions and implementations in one source file, which is awkward. Since we haven't yet learned about separate compilation, we'll have to live with it.

Complete the implementation in accordance with the description of the game. You are allowed to make whatever changes you want to the *private* parts of the classes: You may add or remove private data members or private member functions, or change their types. You must *not* make any deletions, additions, or changes to the *public* interface of any of these classes — we're depending on them staying the same so that we can test your programs. You can, of course, make changes to the *implementations* of public member functions, since the callers of the function wouldn't have to change any of the code they write to call the function. You must **not** declare any public data members, nor use any global variables other than the global constants already in the skeleton code, except that you may add additional global constants if you wish. You may add additional functions that are not members of any class. The word `friend` must not appear in your program.

Any member functions you implement should never put an object into an invalid state, one that will cause a problem later on. (For example, bad things could come from placing a zurt outside the arena.) If a function has a reasonable way of indicating failure through its return value, it should do so. Constructors pose a special difficulty because they can't return a value. If a constructor can't do its job, we have it write an error message and exit the program with failure by calling `exit(1);`. (We haven't learned about throwing an exception to signal constructor failure.)

What you will turn in for this assignment is a zip file containing this one file and nothing more:

1. A text file named **zurts.cpp** that contains the source code for the completed C++ program. This program must build successfully, and its correctness must not depend on undefined program behavior. Your program must not leak memory: Any object dynamically allocated during the execution of your program must be deleted (once only, of course) by the time your main routine returns normally.

Notice that you do not have to turn in a report describing the design of the program and your test cases.

The turn in Project 7 link on the class webpage enables you to turn in your zip file electronically. Turn in the file by the due time above.