**Gamigo US Programming Test**

Please send back your solutions as C or C++ source files attached to your email via zip file. Do not paste your answers into this document or into the body of your email.

You may use any reference materials (books, internet sites, etc.). We strongly recommend that you compile and test your answers. If you write any test code, please include it with your answers.

Write your answers in C or C++ assuming they will become part of a large game codebase. Your code will be called frequently during gameplay, and will be maintained and extended by the entire programming team.

**1.** Write a function that takes an unsigned integer as input, and returns true if all the digits in the base 10 representation of that number are unique.

bool AllDigitsUnique(unsigned int value)

Example:

AllDigitsUnique(48778584) returns false

AllDigitsUnique(17308459) returns true

**2.** Write a function that modifies an input string, sorting the letters according to a sort order defined by a second string. You may assume that every character in the input string also appears somewhere in the sort order string. Make your function as fast as possible for long input strings.

void SortLetters(char\* input, char\* sortOrder)

Example:

char\* inputStr = strdup(“trion world network”);

SortLetters(inputStr, “ oinewkrtdl”);

would modify the input string to “  oooinnewwkrrrttdl”.

**3.** Imagine a maze of interconnected rooms:

-          Each room has a unique name, and can be connected to between zero and four other rooms through doors named north, south, east, and west.

-          Rooms aren’t necessarily spatially coherent. If A is north of B, B might not be south of A.

-          Doors aren’t necessarily bidirectional. If A can be reached from B, B might not be reachable from A.

-          Rooms might connect to themselves.

**3a.** Write a class or struct declaration that shows the structure of a Room. Include a constructor (for classes) or initialization function (for structs).

**3b.** Write a function to connect a new Room to an existing Room.

**3c.** Write a function to determine if any path exists between a starting room and an ending room with a given name. You may add additional members and methods to your Room class or struct as needed. You don’t need to worry about finding the shortest path. It’s only necessary to determine if *any* path exists.

 Use this prototype:

bool Room::PathExistsTo(const char\* endingRoomName);

**4.** Design an elevator controller for a building of ten floors (including the ground floor).

You will need to complete the supplied ElevatorMotor class by writing implementations of the given methods, getCurrentFloor, getCurrentDirection, setCurrentDirection and step. Your ElevatorController class should implement the interface in ElevatorControllerBase and should defer to the completed ElevatorMotor class for all movement and tracking of position.

Design your controller to achieve the following goals (listed in order of priority, first being highest priority) to minimize user irritation and/or ensure proper function:

1. At no time should an idle elevator be unresponsive to summon calls or when a floor button is pressed.
2. At no time should a running elevator car attempt to go below or above the lowest, highest floor respectively.
3. For an embarking user, the maximum time for the user to pick a destination floor after boarding the elevator car is 5 seconds. After which, the car can switch to moving.
4. For an individual user, minimize the time between boarding an elevator car and reaching the destination floor and disembarking.
5. For an individual user, minimize the time between summoning an elevator car and embarking.

#include <unordered\_set>

#include <functional>

namespace Gamigo {

class IFloorEventListener

{

public:

virtual void floorReached(int) = 0;

};

class ElevatorMotor

{

private:

std::unordered\_set<IFloorEventListener\*>

      \_floorReachedEventListeners;

public:

enum Direction { Up = 1, None = 0, Down = -1 };

Direction getCurrentDirection() const;

void setCurrentDirection(Direction direction);

int getCurrentFloor() const;

// Advances the simulation by deltaSeconds seconds

virtual void step(float deltaSeconds);

void addFloorReachedListener(IFloorEventListener\* listener)

{

\_floorReachedEventListeners.insert(listener);

}

void removeFloorReachedListener(

               IFloorEventListener\* listener)

{

\_floorReachedEventListeners.erase(listener);

}

// called when the car has reached a particular floor

virtual void notifyReachedFloor(int floor)

{

for (auto listener : \_floorReachedEventListeners)

{

listener->floorReached(floor);

}

}

};

class ElevatorControllerBase

{

protected:

ElevatorMotor\* \_elevatorMotor;

public:

ElevatorControllerBase(ElevatorMotor\* elevatorMotor):

\_elevatorMotor(elevatorMotor)

{}

// called when an up or down button was pushed on a floor

virtual void summonButtonPushed(

                int summoningFloor,

                ElevatorMotor::Direction direction) = 0;

// called when a button for a floor is pushed inside the car virtual void floorButtonPushed(int destinationFloor) = 0;

};

}