UML

Answering Questions:

Question:

How could you design your system (or modify your existing design) to allow for some generated blocks to disap- pear from the screen if not cleared before 10 more blocks have fallen? Could the generation of such blocks be easily confined to more advanced levels?

Our team can add an extra field in the Block to count the number of rounds that the block survives. If the block survives 10 rounds, then we need to remove the block (delete its address). The cells that point to it will become nullptr and we need to add new method in playBoard to update (blocks above move down by one unit). If in advanced levels there are rules that making blocks disappear if not cleared before 10 more blocks have fallen, the same functions are applied on it. However, if in advanced levels, there are rule that making blocks disappear regardless of rounds, each time when the field that count the number of rounds reaches 10, the class Cell should also get the block’s level. If the level is in 3 or 4, the new method in playBoard to update will not be called and the block should not be deleted.

Question: (Helen)

How could you design your program to accommodate the possibility of introducing additional levels into the system, with minimum recompilation?

We use factory method to implement levels. When we introduce additional levels into the system, we create additional level classes (level i) that implement the Level interface. Each level i class has methods to generate block based on the level. Therefore, when introducing new levels, we only need to recompile the level class.

Question: (Melissa)

How could you design your program to allow for multiple effects to applied simultaneously? What if we invented more kinds of effects? Can you prevent your program from having one else-branch for every possible combination?

Question: How could you design your system to accommodate the addition of new command names, or changes to existing command names, with minimal changes to source and minimal recompilation? (We acknowledge, of course, that adding a new command probably means adding a new feature, which can mean adding a non-trivial amount of code.) How difficult would it be to adapt your system to support a command whereby a user could rename existing commands (e.g. something like rename counterclockwise cc)? How might you support a “macro” language, which would allow you to give a name to a sequence of commands? Keep in mind the effect that all of these features would have on the available shortcuts for existing command names.

Level: Factory Method:

Field:

Class Level0 {

Int getLevel() override { return 0; }

};

Class Block {

Shared\_ptr<CellBoard> cb;

Std::map<std::pair<int, int>, bool> m = { {make\_pair{0, 0}, true}, {make\_pair{1, 0}, true}, … }

Void notifyCB() {

For key in m {

(Cd->theCellBoard)[key.first][key.second].notify(this);

}

}

Bool getState(std::pair<int, int> p) { return m.find(p).second; }

Void up();

Void down();

Void clockwise();

Void counterclockwise();

};

Class CellBoard {

Std::vector<std::vector <Cell> > theCellBoard;

};

Class Cell {

Int x, y; //coordinates

Std::shared\_ptr<Block> b = nullptr;

Bool filled() {

if (b != nullptr) return true;

Else return false;

}

Void notify(Block \*block) {

Std::pair<int, int> coord = make\_pair<x, y>;

If (block -> getState(coord)) b = make\_shared<Block> block;

Else b = nullptr;

}

};