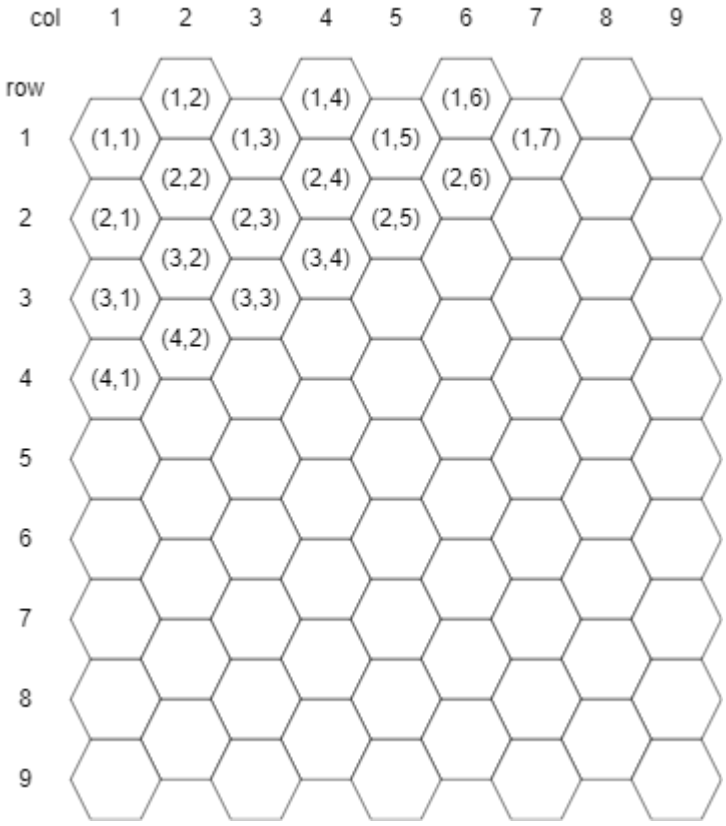
Urban Planning, Balanced Environment, and Adaptive Territory (UPBEAT)

As a newly elected mayor of a newly established city, your job is to build a prospering and sustainable territory. But you need to be committed. A long-term construction plan that can be executed repeatedly will save your budget in the long run, and frequent changes to the construction plan can be costly. Act slow, and your opponent will have an advantage by occupying an unclaimed region. You can either give it up or reclaim it by winning a battle with your opponent.

Overview of UPBEAT

UPBEAT is a turn-based game in which two or more players attempt to claim an entire territory. A territory is an *m*×*n* field of hexagonal regions, where *m* and *n* are as specified in the configuration file. The coordinate of a region is specified by a pair of row and column numbers of the region. Observe that regions on the same row but adjacent columns will skew vertically, as shown in the following figure.



A construction plan is a script that dictates how the city crew will travel from the city center to another region in order to expand the city into a new region, or improve upon a region already

owned by the player. Before the game begins, each player has an opportunity to devise an initial construction plan. Construction plans can be revised during the game, but each player has a time limit on making revisions.

The game ends when the number of players that own any region in the territory is exactly one, i.e., when a player is the only remaining player in the territory.

Setup

First, the game should read the configuration file for necessary parameters.

At the beginning of the game, each player will be given an initial budget as specified in the configuration file, and a starting region randomly selected by the game. This starting region is designated the initial city center for the player.

Each region belonging to a player will have a deposit that can accrue interest. Initially, each player only has the city center, which is given an initial deposit as specified in the configuration file, in addition to the initial budget.

Before the game begins, each player will have an opportunity to devise an initial construction plan, which should be done within the time limit as specified in the configuration file. After the players are done devising the initial construction plan, the timer for construction plan revisions is set for each player. Then, it is the first player's turn to start the game.

Gameplay

At the beginning of each turn, each of the regions belonging to the current player accrues interest, and then the timer for that player resumes counting down. During this time, the player can revise their construction plan. If the player is satisfied with the existing construction plan, they can confirm it, at which point the timer pauses, and the city crew of that player embarks from the city center, performing tasks according to the construction plan until finished. Then, the next player's turn begins.

Interest rates and calculation

An interest rate is specified as a percentage, which is always a positive integer. Although deposit queries in a construction plan can only return integers, UPBEAT should keep track of the deposit in each region as a double, as decimals may affect interest calculations. If d is the current deposit of a region, and r is the interest rate percentage, then the interest for the region in the current turn is $d*r/100$ (floating-point arithmetic), which is then added to the deposit.

If b is the base interest rate percentage as specified in the configuration file, d is the current deposit of a region, and t is the current turn count of a player, the interest rate percentage r to be used is $b * \log_{10} d * \ln t$.

Grammar for construction plans

Construction plans are governed by the following grammar. Terminal symbols are written in `monospace font`; nonterminal symbols are written in *italics*.

Plan → *Statement*+
Statement → *Command* | *BlockStatement* | *IfStatement* | *WhileStatement*
Command → *AssignmentStatement* | *ActionCommand*
AssignmentStatement → `<identifier>` = *Expression*
ActionCommand → `done` | `relocate` | *MoveCommand* | *RegionCommand* | *AttackCommand*
MoveCommand → `move` *Direction*
RegionCommand → `invest` *Expression* | `collect` *Expression*
AttackCommand → `shoot` *Direction Expression*
Direction → `up` | `down` | `upleft` | `upright` | `downleft` | `downright`
BlockStatement → `{` *Statement** `}`
IfStatement → `if (` *Expression* `) then` *Statement* `else` *Statement*
WhileStatement → `while (` *Expression* `)` *Statement*
Expression → *Expression* `+` *Term* | *Expression* `-` *Term* | *Term*
Term → *Term* `*` *Factor* | *Term* `/` *Factor* | *Term* `%` *Factor* | *Factor*
Factor → *Power* `^` *Factor* | *Power*
Power → `<number>` | `<identifier>` | `(` *Expression* `)` | *InfoExpression*
InfoExpression → `opponent` | `nearby` *Direction*

- \+ means at least one
- * means zero or more
- `<number>` is any nonnegative integer literal that can be stored as Java's long data type.
- `<identifier>` is any string not a reserved word.
  Identifiers must start with a letter, followed by zero or more alphanumeric characters.

The following strings are reserved words and cannot be used as identifiers: `collect, done, down, downleft, downright, else, if, invest, move, nearby, opponent, relocate, shoot, then, up, upleft, upright, while`

Construction plan evaluation

Variables

Each variable has the initial value of 0. After each turn, a variable retains its value for the beginning of the next turn. For example, if a variable *x* has been assigned a value of 47 at turn *t*, then its value will remain at 47 at the beginning of turn *t*+1, and could be assigned to another value in the next round of evaluation.

Variables are local, i.e., they are never shared between players. If two players have construction plans containing variables of the same name, their values can be different.

Assignments to a variable *x* in player A's construction plan do not affect *x*'s value in another player's construction plan.

Special variables

UPBEAT designates many identifiers in construction plans as special, read-only variables. An attempt to assign any of these variables results in a "no-op": it's like this assignment never exists. The special variables are as follows.

`rows`

Whenever this variable is evaluated, the number of rows (*m*) in the territory is returned.

`cols`

Whenever this variable is evaluated, the number of column (*n*) in the territory is returned.

`currow`

Whenever this variable is evaluated, the current row number of the city crew is returned.

`curcol`

Whenever this variable is evaluated, the current column number of the city crew is returned.

`budget`

Whenever this variable is evaluated, the player's remaining budget is returned.

`deposit`

Whenever this variable is evaluated, the current deposit on the current region occupied by the city crew is returned. If this region belongs to no players, the current deposit (if any) should be negated before returned. The decimal part of the deposit (if any) is truncated when returned.

`int`

Whenever this variable is evaluated, the interest percentage rate for the current deposit on the current region occupied by the city crew, as calculated according to the specification, is returned.

`maxdeposit`

Whenever this variable is evaluated, the maximum possible deposit for a region, as specified in the configuration file, is returned.

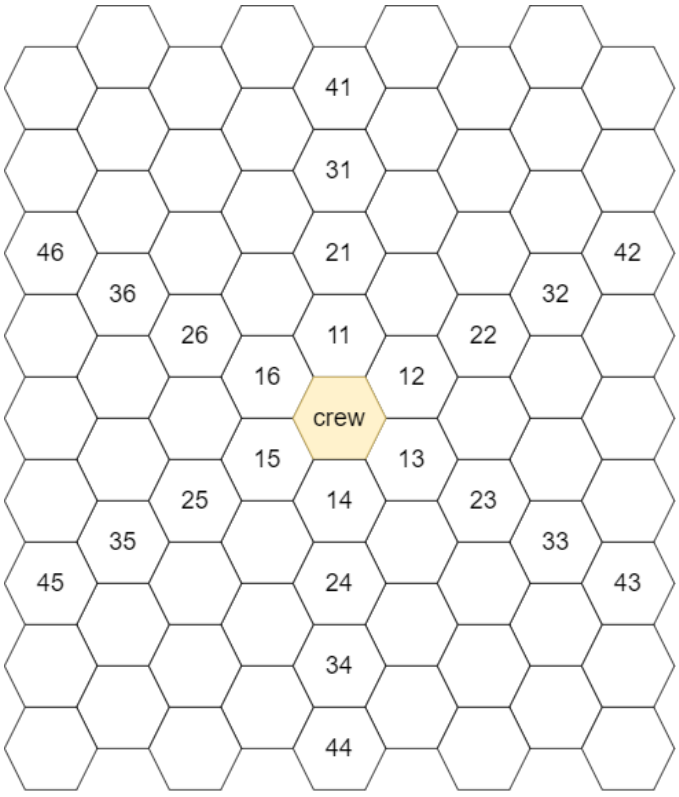
`random`

Whenever this variable is evaluated, a random value between 0 and 999 (inclusive) should be returned.

Information expressions

`opponent` expression

The `opponent` command returns the location of the closest region belonging to an opponent in one of the six directions from the current region occupied by the city crew. The following diagram shows a pattern of the expression's results for the locations to be returned. Blank regions are never reported.



The value of each location is relative to the city crew. The unit digit is between 1 and 6, and indicates the direction of the location. The remaining digits indicate the distance from the city crew to that location.

If there are multiple opponent's regions with the same minimal distance from the city crew, return the location of the one with the lowest direction number. In other words, the returned value is the smallest possible one.

If there are no visible opponent's regions whatsoever from the current location of the city crew, the `opponent` command should return 0.

`nearby` function

The `nearby` function looks for the opponent's region closest to the current location of the city crew in a given direction. This command should return $100x+y$, where x is the distance from the city crew to that region as shown in the diagram above, and y is the number of digits in the current deposit for that region, which can be helpful when planning attacks.

If no opponent owns a region in the given direction, the `nearby` function should return 0.

Action commands

During each turn, a number of action commands can be executed, although certain kinds of commands may be executed at most once per turn. After executing such commands, the turn ends automatically.

`done` command

Once executed, the evaluation of the construction plan in that turn ends. This is similar to the `return` statement in a procedure.

`relocate` command

This command relocates the city center to the current region. The cost to relocate the city center is $5x+10$, where x is the minimum moving distance from the current city center to the destination region (regardless of the presence of any opponent's region in between). This cost is deducted from the player's budget. If the player does not have enough budget to execute this command, or if the current region does not belong to the player, the relocation fails. Once executed (regardless of the outcome), the evaluation of the construction plan in that turn ends.

`move` command

The `move` command moves the city crew one unit in the specified direction. If the destination region belongs to another opponent, the command acts like a no-op. Whenever this command is executed (regardless of validity), the player's budget is decreased by one unit. If the player does not have enough budget to execute this command, the evaluation of the construction plan in that turn ends.

`invest` command

The `invest` command adds more deposits to the current region occupied by the city crew. The total cost of an investment is $i+1$, where i is the investment amount. If the player does not have enough budget to execute this command, the command acts like a no-op, but the player still pays for the unit cost of the command. Otherwise, the player's deposit in the current region is increased by i, but will not exceed the maximum deposit as specified in the configuration file. A player may invest in a region belonging to no player as long as that region is adjacent to another region belonging to the player.

`collect` command

The `collect` command retrieves deposits from the current region occupied by the city crew. Whenever this command is executed, the player's budget is decreased by one unit. If the player does not have enough budget to execute this command, the evaluation of the construction plan in that turn ends. If the specified collection amount is more than the deposit in the current region, the command acts like a no-op, but the player still pays for the unit cost of

the command. Otherwise, the player's deposit in the current region is decreased by the collection amount, which is then added to the player's available budget. If the deposit becomes zero after the collection, the player loses the possession of that region.

`shoot` command

The `shoot` command attempts to attack a region located one unit away from the city crew in the specified direction.

Each attack is given an expenditure, i.e., how much the player would like to spend in that attack. This will be deducted from the budget. Each attack also has a fixed budget cost of one unit. That is, the total cost of an attack is $x+1$, where x is the given expenditure. If the player does not have enough budget to execute this command, the command acts like a no-op. Otherwise, the opponent's deposit in the target region will be decreased no more than the expenditure. Specifically, if the expenditure is x and the opponent's deposit is d, then after the attack, the deposit will be $\max(0, d-x)$. If the deposit becomes less than one, the opponent loses ownership of that region. In this case, the target region does not automatically become the current player's region; the player needs to move into that region and invest in order to claim it.

If the target region is unoccupied, the player still pays the cost of the attack, but the attack itself will have no effects otherwise.

If the target region belongs to the current player (i.e., a player is attacking its own region), the command has the same effect as the normal scenario, i.e., it acts as self destruction. Be careful who you shoot at!

Finally, if the target region is a city center, and the attack reduces its deposit to zero, the attacked player loses the game. The other regions that belong to the losing player will be ownerless, but the deposit will remain there for the remaining players to discover and claim it. Ownerless deposits will not accrue interests.

Arithmetic evaluation

Evaluation of expressions in construction plans uses integer arithmetic. Division using the `/` operator is integer division, truncating decimals. Overflow and underflow are allowed to occur in order to fit results in Java's long data type.

Boolean evaluation

Conditions in the `if` and `while` statements evaluate to integers. To interpret integer values as booleans, positive values are considered true, and negative values and zero are considered false.

Avoiding infinite loops

It is possible for a `while` loop to execute for too many iterations. To avoid infinite loops during an evaluation of a construction plan, a `while` loop that has run for 10000 iterations is terminated, i.e., its guard automatically becomes false, and the subsequent statement is then evaluated.

In effect, a `while` statement in a construction plan

```
while (e) s
```

works like the following code in most programming languages:

```
for (int counter = 0; counter < 10000 && e > 0; counter++) s
```

Sample construction plan

`#` indicates the beginning of a comment, and is not part of the construction plan. You can choose to support end-of-line comments in your tokenizer if desired.

```
t = t + 1  # keeping track of the turn number
m = 0  # number of random moves
while (deposit) { # still our region
  if (deposit - 100)
  then collect (deposit / 4)  # collect 1/4 of available deposit
  else if (budget - 25) then invest 25
  else {}
  if (budget - 100) then {} else done  # too poor to do anything else
  opponentLoc = opponent
  if (opponentLoc / 10 - 1)
  then  # opponent afar
    if (opponentLoc % 10 - 5) then move downleft
    else if (opponentLoc % 10 - 4) then move down
    else if (opponentLoc % 10 - 3) then move downright
    else if (opponentLoc % 10 - 2) then move right
    else if (opponentLoc % 10 - 1) then move upright
    else move up
  else if (opponentLoc)
  then  # opponent adjacent to city crew
    if (opponentLoc % 10 - 5) then {
      cost = 10 ^ (nearby upleft % 100 + 1)
      if (budget - cost) then shoot upleft cost else {}
    }
    else if (opponentLoc % 10 - 4) then {
      cost = 10 ^ (nearby downleft % 100 + 1)
      if (budget - cost) then shoot downleft cost else {}
    }
    else if (opponentLoc % 10 - 3) then {
      cost = 10 ^ (nearby down % 100 + 1)
      if (budget - cost) then shoot down cost else {}
    }
```

```
    else if (opponentLoc % 10 - 2) then {
      cost = 10 ^ (nearby downright % 100 + 1)
      if (budget - cost) then shoot downright cost else {}
    }
    else if (opponentLoc % 10 - 1) then {
      cost = 10 ^ (nearby upright % 100 + 1)
      if (budget - cost) then shoot upright cost else {}
    }
    else {
      cost = 10 ^ (nearby up % 100 + 1)
      if (budget - cost) then shoot up cost else {}
    }
  else {  # no visible opponent; move in a random direction
    dir = random % 6
    if (dir - 4) then move upleft
    else if (dir - 3) then move downleft
    else if (dir - 2) then move down
    else if (dir - 1) then move downright
    else if (dir) then move upright
    else move up
    m = m + 1
  }
}  # end while
# city crew on a region belonging to nobody, so claim it
if (budget - 1) then invest 1 else {}
```

Display and controls

A territory may have a lot of regions. Sometimes, it might not be possible to show every region and have you focus on some part of the territory. Therefore, you should be able to zoom in and out parts of the territory. Even if a player is looking at a certain part of the territory, that doesn't mean the other parts stop working.

We recommend that you display the territory on a webpage. As coders, you can use the Spring Framework to link your Java backend to your web frontend. As players, you should be able to send commands from the frontend to the Java backend.

Think about how to display the territory so that players have enough information to decide what to do and enjoy the game.

Configuration file

The configuration file contains many parameters that can be adjusted to balance the game. Each parameter value must be representable in Java's long data type.

Each parameter specification is of the form `<name>=<value>`. The meanings of parameter names are as follows:

- `m`: the number of rows in the territory
- `n`: the number of columns in the territory
- `init_plan_min`: the number of minutes allowed for the initial construction plan specification
- `init_plan_sec`: the number of seconds allowed for the initial construction plan specification (between 0 and 59 inclusive)
- `init_budget`: the initial budget
- `init_center_dep`: the initial city center deposit
- `plan_rev_min`: the number of minutes allowed for the construction plan revisions
- `plan_rev_sec`: the number of seconds allowed for the construction plan revisions (between 0 and 59 inclusive)
- `rev_cost`: construction plan revision cost
- `max_dep`: maximum deposit per region
- `interest_pct`: interest rate percentage

The order of names in the configuration file need not follow the list above.

Sample configuration file

```
m=20
n=15
init_plan_min=5
init_plan_sec=0
init_budget=10000
init_center_dep=100
plan_rev_min=30
plan_rev_sec=0
rev_cost=100
max_dep=1000000
interest_pct=5
```

Project timeline (subject to change)

- Wed 24 Jan:
    - Design overview document for construction-plan evaluator due
        - This includes the outline of class hierarchy for your parser and evaluator.
        - Your design will be given feedback, so you can address flaws before you submit your code.
- Wed 7 Feb:
    - Construction-plan evaluator due
        - At this point, your project should be able to parse and execute construction plans, given a mock game state required for the execution.
    - Design overview document for game state and user interface
        - This includes the outline of the display (view and controller) and your class hierarchy (model).
        - Your design will be given feedback, so you can address flaws before you submit your code.
- Wed 21 Feb:
    - Implementation of game state and user interface due
        - At this point, your evaluator should be integrated with the game state.
        - At this point, your user interface should be displayable and can mock some construction commands
    - Design overview document for server-client implementation
        - This includes the outline of the interaction between game state and user interface, including interaction between modules from other groups.
        - Your design will be given feedback, so you can address flaws before you submit your code.
- Wed 6 Mar:
    - Server-client implementation due
        - At this point, your game state should be able to interact with user interfaces from other groups.
        - At this point, your user interface should be able to interact with game state from other groups.
    - Project report due
        - Architectural overview of your system
        - Report on what was done according to plan and outside the plan
        - Retrospect on what was done well, and what could have been better
    - Possible extension: after the final exam period (no later than Wed 27 Mar)