All time in PST

	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
Kyle	Available 6-10pm	Available 6-10pm	Available 6-10pm	Available 6-10pm	Available 6-10pm	Available all day	Available all day
Leah	Available daytime till 2pm/after 6pm	Available daytime till 2pm/after 6pm	Available daytime till 12pm/after 6pm	Available daytime till 3jpm/after 6pm	Available daytime till 3pm	Not available	Not available
Kukrim	Available 6:20-10pm	Available 6-10pm	Available 6-10pm	Available 6-10pm	Available 6-10pm	Available all day	Available all day
Serena	Available 6pm-12am	Available 6pm-12am	Not available	Available 6pm-12am	Available all day	Available 12pm - 5:30pm	Available all day

Meetings scheduled:
Monday - 6:30pm PST
Tuesday - 6:30pm PST
Wednesday - 6:30pm PST
Thursday - 6:30pm PST
Friday - 6:30pm PST

Action items before Monday meeting:
Serena and Leah - jumpstart critter class member vars and functions
Kukrim - set up repl.it for the team

Notes from Monday:
1. **Doodlebugs move first, then ant moves.**
2. **Ant does not move into doodlebug's space**
3. **Use Critter::getSpace() to obtain character associated with the space**
4. **World::checkCritter() will check space we are trying to move to**

Design:
As of Monday 1:30pm - Leah has updated the program files and uploaded version 3 of files to canvas as a zip file containing 10 files (ant.hpp ant.cpp critter.hpp critter.cpp doodlebug.hpp doodlebug.cpp world.hpp world.cpp main.cpp makefile). The code currently creates a board of critter pointers and objects, assigns one and and one doodlebug to the board and prints the board for the number of steps the user enters. Updates she has made on the design are in this document in blue. She will present the files to the group tonight at our meeting.

Design questions that came up while Leah was working on the project Monday morning:

- Board needs to be a 2D board of critter pointers

        - How do will we keep track of which spots have ant/doodlebugs/empty
        - How do we randomly place the ants/doodle bugs on the board to start the simulation?
            - I have a test version of this in the world.cpp constructor, but it didn't end up working for both ant and doodlebug.
    - I moved steps to the world class from the critter class since it is the representation of time in the game and for one step all the ants and all the doodlebugs on the board will move. I haven't created a oneStep() world class function yet, but if we do this could be where we loop through the board to call the move function.

Critter class
protected:
Int xLocation
Int yLocation
Char onSpace;

Public:
getXLocation ();
setXLocation();
getYLocation ();
setYLocation()
Virtual char printSpace() - how we know what critter is on what space
Virtual void move (); - move is different for ant than doodlebug so the critter func is empty
Virtual void breed (); //question can a function within a class call the constructor of the class?

Ant class
Public:
Ant() sets onSpace = 'O'; seeds random number generator
~Ant()

Doodlebugs class
Private:
Int starveDays;
Int steps;

Public:
Doodlebug() sets onSpace = 'X'; seeds random number generator
~Doodlebug()
Void starve();

World class
Private:
    Int rowCount

```
        Int columnCount;
        Int numAnts;
        Int numDoodlebugs;
        Int stepCount;
        Critter ***board = nullptr;
Public:
```
World() - I used the constructor to create the 2D board since each world only has one board (The project description says this needs to be an array of critter pointers

~World() - this deletes the board array of critter pointers

```
Void setSteps(int);
Int getSteps();
Void printBoard();
```

Running the program functions

randomFunction(); // From leah - so far I have just been putting rand() inside the classes as needed rather than creating a specific function for this. We can change this if we want.

oneStep(); // have not added this one yet

Menu class

//use prior project/assignment class

Monday update from Leah - I used menu functions from previous assignments in the main.cpp rather than create a new class.

Main.cpp

```
void startMenu(int &);
int getInteger(string);

int main()
{
    int timeSteps = 0;
    startMenu(timeSteps);

    World test;
    test.setSteps(timeSteps);

    do{
        test.printBoard();
    }while(test.getSteps() > 0);

void startMenu(int &steps)
{
    int minSteps = 1;
```

```
	int maxSteps = 20;
	int maxRow = 20;
	int minRow = 2;

	cout << "Welcome to Predator and Prey"  << endl;
	cout << "The length of this game is measured in time steps." << endl;
	cout << "Time steps are positive integers only" << endl;

	// Get user input for number of time steps, validate input
	steps = getInteger("How many time steps would you like in this game?");
	while(steps > maxSteps || steps < minSteps)
	{
		cout << "The minimum number of steps is " << minSteps << "." << endl;
		cout << "The maximum number of steps is " << maxSteps << "." << endl;
		steps = getInteger("Please enter a valid integer for the number of steps in game");
	}
}


int getInteger(string prompt)
{
	string line = "";
	int value;
	int digits = 0;
	bool isValid;
	cout << prompt << endl;

	do{
		getline(cin, line);
		for(int i = 0; i < line.length(); i++)
		{
			if(isdigit(line[i]))
			{
				digits += 1;
			}
		}
		if(digits == line.length() && !(line.length() == 0))
		{
			isValid = true;
		}
		else
		{
			cout << "Invalid entry. Please enter an integer." << endl;
```

```
                isValid = false;
            }
            digits = 0;
    }while(!(isValid));

    value = stoi(line);
    line = "";
    return value;
}
```

Makefile:

```
PredatorPrey: main.o world.o doodlebug.o critter.o ant.o
    g++ -std=c++0x -g main.o world.o doodlebug.o critter.o ant.o -o PredatorPrey

main.o: main.cpp
    g++ -std=c++0x main.cpp -c

world.o: world.cpp
    g++ -std=c++0x world.cpp -c

doodlebug.o: doodlebug.cpp
    g++ -std=c++0x doodlebug.cpp -c

critter.o: critter.cpp
    g++ -std=c++0x critter.cpp -c

ant.o: ant.cpp
    g++ -std=c++0x ant.cpp -c

clean:
    rm *.o PredatorPrey
#end of Makefile
```

Work distribution: 5 pair programming sessions lasting approximately 2 hours. Additionally, Kukrim, Serena, and Leah all worked collaboratively on code as their schedules allowed. Kyle was responsible for the testing table and the reflection document.

Test Plan:

Test Table:

Test Case	Input	Driver Functions	Expected Output	Actual Output

Input too Low	-1	Int main()	Program should throw error message	Program throws error message “Invalid entry. Please enter an integer.”
Input at 0	0	Int main()	Program should throw error message and promp user to re-input integer.	Program throws error message “The minimum number of steps is 1. The maximum number of steps is 20. Please enter a valid integer for the number of steps in the game.”
Input in Correct Range	5	Int main()	Program should run as expected.	Program runs as expected
Input Extreme Low	-3333	Int main()	Program should throw error message and promp user to re-input integer.	Program throws error message “Invalid entry. Please enter an integer.”
Input Extreme High	999999999	Int main()	Program should throw error message and promp user to re-input integer.	Program throws error message “The minimum number of steps is 1. The maximum number of steps is 20. Please enter a valid integer for the number of steps in the game.”
Input Wrong Data Type	ttt	Int main()	Program should throw error message and promp user to re-input integer.	Program throws error message “Invalid entry. Please enter an integer.”
Input in Correct Range	5	Int main()	Program should spawn 100 ants and 5 doodlebugs. Ants should spawn after 3 turns.	Programs behaves as expected. Initially the board starts with 100 ants and 5 doodlebugs. 6 Ants were eaten in the first two three turns so the number of Ant objects after breeding is 181. This is because there were 3 ants that could not find a open space to breed. Additionally, we saw the expected behavior of a Doodlebug object starving to death after turn 4 because it had not eaten in 3 turns.
Input in Correct Range	5, 5	Int main()	Program executes as expected and continues using the same board when the 2nd round of time steps is entered.	Program executes as expected and runs another iteration when given a 2nd time step duration. All data memebers are kept from the first iteration and the program continues to run on the results accumulated from the original time step in the new time step duration.

Reflection:

Our group had a few issues where our design did not translate well to execution. Please find the following examples are the learning opportunities created by these difficulties.

Move Function:
The first design our group had for the Move function was to pass a pointer to a the starting position and the ending position, as well as the random number that was generated for determining which direction the Ant or Doodlebug should be moving. We originally had difficulty with this design because we were not passing the pointers into the Move function by reference so we were manipulating a local copy of the object and the changes were not being reflected when the board was subsequently printed after the time step finished. We updated the design for the Move functions to pass the entire 2D array of Critter pointers to the Ant and Doodlebug class Move functions as well as the current location in the 2D array as we are looping through to identify and move all Doodlebug and Ant objects. Overall this was a large departure from our original strategy of passing two pointers by reference but found this solution to be much more flexible.

Edge Case Logic:
Our program was giving us an out of bounds error when we were initially testing the logic we wrote for the move functions. We were receiving these errors because we had not included logic in our move functions to handle edge cases. Our group had accidentally introduced a logical error into our program that allowed us to jump the bounds of the array because we were focused on getting the values working in the middle of the array for our move, breed, and starve functions. After a number of rounds of debugging, Kukrim identified that we had introduced a bug into our program by artificially starting our loop to iterate through all of the spaces in the 2D array at the 1 space rather than the 0 space. After realizing our issue, we updated conditional logic to start the iteration at the correct place and updated some conditional logic for moving Ants and Doodlebugs to handle edge cases and our program worked as expected. We learned as a group that we need to do a better job of documenting when are where we update the logic of the program for a specific purpose like debugging or testing and to make sure that you change the logic back when you get the desired understanding of the flow of the program.

Breed Function Issue 1:
The first issue our group had with the Breed function was the we were calling the breed function after an object had been moved to its new location but were still passing the original location of the object (Ant or Doodlebug) to the Breed function. This was causing the program to think that the object was not eligible to Breed and no new objects were getting initialized. After realizing this logical error the program was updated to pass the new location of the object to the Breed function and if the object had met the age criteria, a new object of that type was created in a random and unoccupied adjacent space.

Breed Function Issue 2:
The second issue our group had with the Breed function was that we did not encapsulate the Breed function in a while loop so the breed function was only randomly checking one direction and if it was occupied no other direction was being considered and no breeding activity was taking place. After identifying that this was an issue, the logic was updated to encapsulate the breed function call in a do while loop that only terminated after a new Ant of Doodlebug object was placed on the board in a random and unoccupied adjacent cell.

Lack of Data Member for Tracking Survival:
Finally, our group had an issue in tracking survival of each of the Doodlebug objects. We had trouble communicating which type of step the object should be tracking to calculate the passage of time that is considered for starvation of the Doodlebug object. Eventually we realized that we were all using different language to measure the same piece of data. We ended up creating a data member to track how many time steps had been taken by an object. This allowed us to determine when a Doodlebug object had starved because it hadn’t eaten an ant object in the correct number of turns. This issue illuminated the need for clear and concise communication and a standard method for description data members/components of a program in a group as there are many different ways that data members of components of programs can be described. This causes confusion if you’re not using a shared terminology as a group.