Lyon Kee

keel@oregonstate.edu

Project #2: Functional Decomposition

Key snippets of code

```
77  void
78  WaitBarrier( )
79  {
80      omp_set_lock( &Lock );
81      {
82          NumAtBarrier++;
83          if( NumAtBarrier == NumInThreadTeam )
84          {
85              NumGone = 0;
86              NumAtBarrier = 0;
87              // let all other threads get back to what they were doing
88              // before this one unlocks, knowing that they might immediately
89              // call WaitBarrier( ) again:
90              while( NumGone != NumInThreadTeam-1 );
91              omp_unset_lock( &Lock );
92              return;
93          }
94      }
95      omp_unset_lock( &Lock );
96
97      while( NumAtBarrier != 0 ); // this waits for the nth thread to arrive
98
99      #pragma omp atomic
100     NumGone++;          // this flags how many threads have returned
101 }
```

This code sets a lock and waits for all threads to reach the barrier before proceeding to allow us to ensure the values are synchronized between each section of computation, assignment, and print/update of env.

```
113 void Deer() {
114     int prevZombie = 1;
115     while (NowYear < 2030 ){
116         int nextNumDeer = NowNumDeer;
117         int carryingCapacity = (int)( NowHeight );
118         // printf("subtracted %d deers\n", NowNumZombie - prevZombie);
119         nextNumDeer -= NowNumZombie - prevZombie;
120         prevZombie = NowNumZombie;
121         if( nextNumDeer < carryingCapacity )
122                 nextNumDeer++;
123         else
124                 if( nextNumDeer > carryingCapacity )
125                         nextNumDeer--;
126
127         if( nextNumDeer < 0 )
128                 nextNumDeer = 0;
129
130
131         WaitBarrier( ); // DoneComputing barrier:
132
133         NowNumDeer = nextNumDeer;
134         WaitBarrier( ); // DoneAssigning barrier:
135
136
137         WaitBarrier( ); // DonePrinting barrier:
138     }
139 }
```

In the computation stage, we calculate the next value for the coming month and save it as a temporary variable, this is because we don't want to write to the memory that holds the current value as other threads are also using that to compute next values. On the assignment stage we set to it because other threads are not setting or reading it allowing us to avoid reading/setting wrong values.

```
void Grain() {
    while (NowYear < 2030 ){
        float tempFactor = exp( -SQR(( NowTemp - MIDTEMP ) / 10. ));
        float precipFactor = exp( -SQR(( NowPrecip - MIDPRECIP ) / 10. ));
        float nextHeight = NowHeight;
        nextHeight += tempFactor * precipFactor * GRAIN_GROWS_PER_MONTH;
        nextHeight -= (float)NowNumDeer * ONE_DEER_EATS_PER_MONTH;
        if( nextHeight < 0. ) nextHeight = 0.;
        WaitBarrier( ); // DoneComputing barrier:

        NowHeight = nextHeight;
        WaitBarrier( ); // DoneAssigning barrier:


        WaitBarrier( ); // DonePrinting barrier:
    }
}
```

In the computation stage, we calculate the next value for the coming month and save it as a temporary variable, this is because we don't want to write to the memory that holds the current value as other threads are also using that to compute next values. On the assignment stage we set to it because other threads are not setting or reading it allowing us to avoid reading/setting wrong values.

```
void MyAgent() {
    while (NowYear < 2030 ){
        int NextNumZombie = NowNumZombie;
        int TempDeer = NowNumDeer;
        if (NowNumDeer) {
            for (int i = 0; i < NowNumZombie && TempDeer; ++i) {
                if ((rand() % (int) (1 / ZOMBIE_CHANCE_TO_INFECT)) == 0) {
                    ++NextNumZombie;
                    --TempDeer;
                }
            }
        }
        WaitBarrier( ); // DoneComputing barrier:

        NowNumZombie = NextNumZombie;
        WaitBarrier( ); // DoneAssigning barrier:


        WaitBarrier( ); // DonePrinting barrier:
    }
}
```

In the computation stage, we calculate the next value for the coming month and save it as a temporary variable, this is because we don't want to write to the memory that holds the current value as other threads are also using that to compute next values. On the assignment stage we set to it because other threads are not setting or reading it allowing us to avoid reading/setting wrong values.

```
167 void Watcher() {
168     while (NowYear < 2030 ){
169         WaitBarrier( ); // DoneComputing barrier:
170 
171         WaitBarrier( ); // DoneAssigning barrier:
172 
173         printf("%16d, %16d, %16d, %16f, %16f, %16f, %16d, %16d\n", NowYear, NowMonth, (NowYear-2024)*12+NowMonth, inches_cm(NowPrecip), F_C(NowTemp), inches_cm(NowHeight)
174 
175         float ang = (  30.*(float)NowMonth + 15.  ) * ( M_PI / 180. );  // angle of earth around the sun
176 
177         float temp = AVG_TEMP - AMP_TEMP * cos( ang );
178         NowTemp = temp + Ranf( -RANDOM_TEMP, RANDOM_TEMP );
179 
180         float precip = AVG_PRECIP_PER_MONTH + AMP_PRECIP_PER_MONTH * sin( ang );
181         NowPrecip = precip + Ranf( -RANDOM_PRECIP, RANDOM_PRECIP );
182         if( NowPrecip < 0. ) NowPrecip = 0.;
183 
184         if (++NowMonth == 12) { // update time environment
185             NowMonth = 0;
186             ++NowYear;
187         }
188 
189 
190         WaitBarrier( ); // DonePrinting barrier:
191     }
192 }
```

We wait for the assigning barrier to finish assigning, then we print out all values to ensure that we are reading the correct values after all threads have written to it.

```
243     InitBarrier( NUMT );
244 
245     #pragma omp parallel sections
246     {
247         #pragma omp section
248         {
249             Deer();
250         }
251 
252         #pragma omp section
253         {
254             Grain();
255         }
256 
257         #pragma omp section
258         {
259             Watcher();
260         }
261 
262         #pragma omp section
263         {
264             MyAgent();
265         }
266     }
```

Line 243: we are initializing 4 threads, one for each agent.

Line 245: we are starting the 4 threads as sections

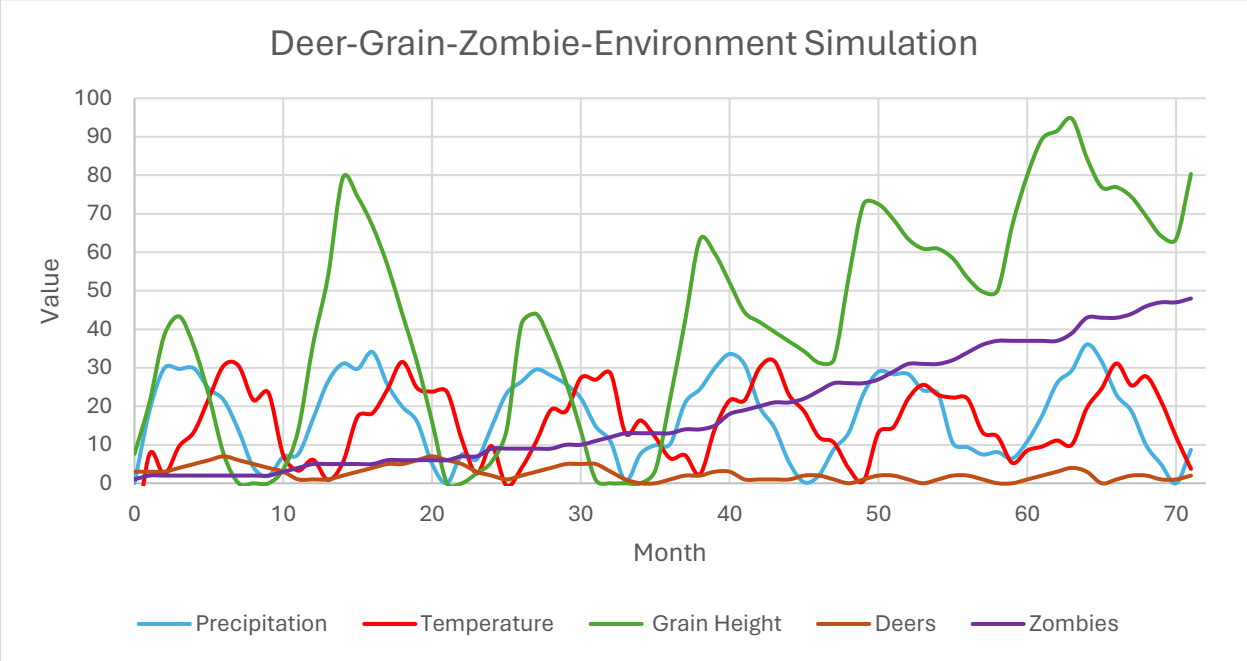
Line 247: We are starting each section and proving an agent to each, this is done 4 times, once for each section.

Tables of data (monthly statistics of the simulated environment)

NumMonths	Precipitation(cm)	Temperature(C)	Grain Height(cm)	Deer	Zombies
0	0	-17.7778	7.620001	3	1
1	18.69234	7.524721	20.90352	3	2
2	29.855	2.212626	38.43515	3	2
3	29.69721	9.593156	43.363	4	2
4	29.82199	13.36292	35.45021	5	2
5	24.57088	22.08142	22.75149	6	2
6	21.56438	30.57476	7.511494	7	2
7	13.68571	30.47162	0	6	2
8	4.106534	21.5881	0	5	2
9	1.917792	23.52018	0	4	2
10	6.90206	7.296147	3.61998	3	3
11	7.532763	3.279813	13.78407	1	4
12	16.7457	6.104012	36.06644	1	5
13	26.37089	0.870355	53.64672	1	5
14	31.07486	5.371352	79.3066	2	5
15	29.72627	17.3546	74.36031	3	5
16	34.07946	18.20991	66.79879	4	5
17	25.60821	24.38951	56.63887	5	6
18	19.87765	31.53747	43.93887	5	6
19	15.91152	24.71269	31.23891	6	6
20	5.049041	23.80409	15.999	7	6
21	0	23.73308	0	6	6
22	7.487213	11.32389	0	5	7
23	6.216679	2.535409	2.611294	3	7
24	14.65241	9.650523	5.58101	2	9
25	23.29818	-0.61912	13.6915	1	9
26	26.36007	3.885852	41.28184	2	9
27	29.54647	10.8712	43.98682	3	9
28	27.95143	19.10752	36.39528	4	9
29	25.71904	18.73299	26.27613	5	10
30	22.08731	27.37793	13.57613	5	10
31	14.74997	26.90554	0.876131	5	11
32	10.76102	28.51713	0	3	12
33	0.593882	12.99216	0	1	13
34	7.578244	16.32514	0	0	13
35	9.867494	11.95937	3.364759	0	13
36	10.18605	6.478765	21.98435	1	13
37	20.85483	7.29493	42.13133	2	14
38	24.45636	2.310486	63.31398	2	14
39	29.91383	14.10386	59.67081	3	15

40	33.56624	21.53349	52.05294	3	18
41	30.8269	21.55777	44.43515	1	19
42	19.92444	29.96891	41.89515	1	20
43	14.4663	31.72457	39.35515	1	21
44	5.36953	22.8755	36.81542	1	21
45	0.264838	18.68254	34.2915	2	22
46	2.072929	11.98734	31.28705	2	24
47	8.829453	10.57066	32.11015	1	26
48	13.15944	3.883945	53.48867	0	26
49	23.31436	0.66273	72.53671	1	26
50	28.9734	13.18294	72.51396	2	27
51	28.31154	14.5383	68.54227	2	29
52	28.35373	21.98783	63.46368	1	31
53	24.14359	25.56975	60.92369	0	31
54	23.22718	22.96314	60.92414	1	31
55	10.44442	22.2495	58.38489	2	32
56	9.366292	21.87521	53.30597	2	34
57	7.477454	13.22707	49.74797	1	36
58	8.091782	12.13049	50.03349	0	37
59	6.461163	5.285759	67.11819	0	37
60	10.87736	8.51331	79.97354	1	37
61	17.59922	9.534641	89.41383	2	37
62	26.00484	11.11426	91.54147	3	37
63	29.35289	10.0455	94.6876	4	39
64	36.0513	19.5221	84.54378	3	43
65	31.62093	24.59548	76.92383	0	43
66	22.94199	31.13249	76.92383	1	43
67	18.71146	25.40098	74.38385	2	44
68	9.795311	27.76756	69.30385	2	46
69	4.909836	21.23242	64.22557	1	47
70	0	12.04499	63.41084	1	47
71	8.685138	3.744032	80.32626	2	48

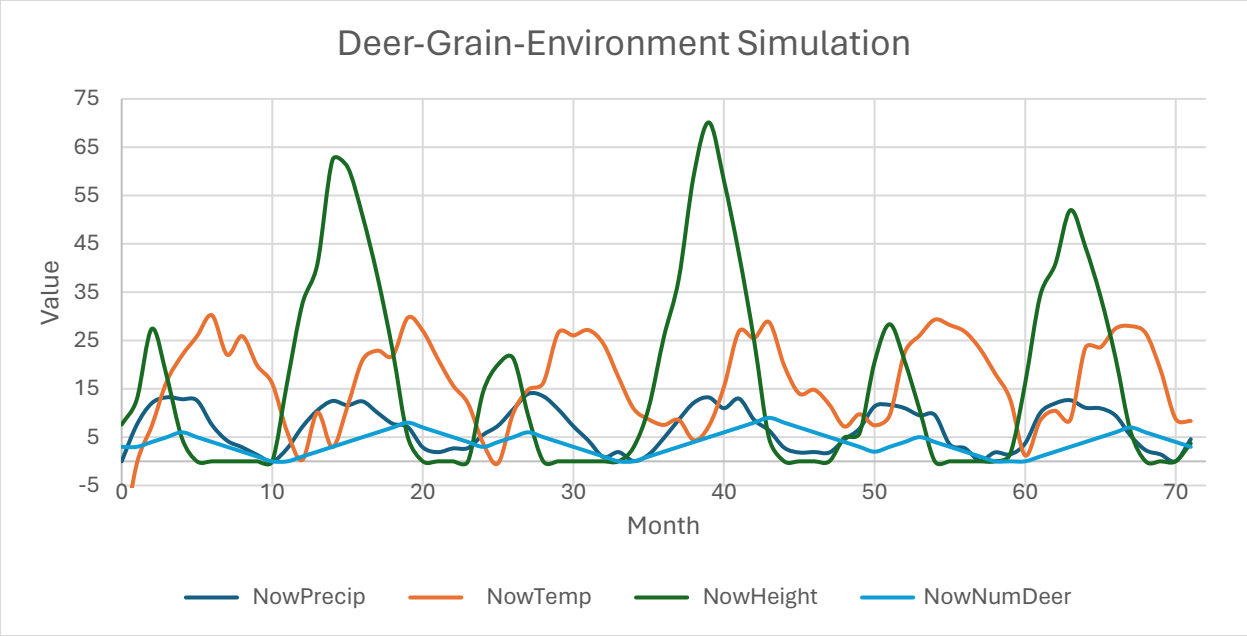
Graphs of data



Units:

- Precipitation
  - cm
- Temperature
  - Celsius
- Grain Height
  - cm
- Deers
  - count
- Zombies
  - count

The above graph is the graph of deer, grain, and zombie. Zombie is the added agent where each zombie is able to infect deer at a chance of 5%. The zombies do not die, and the infected deer is considered a zombie instead of a deer. The deer population we see is the number of deer before infection because it would be 0 if we were to not show that, and it could be confusing to explain why zombies increase at that rate. We observe that zombies increases more when there are more deer birth. We see that the deer population is struggling to survive due to a high number of zombie, making it deadly and very infectious. With no deer to consume grain, we observe that it is able to grow during its season due to lack of "predator" and it slowly decreases out of season, this does not go to 0 because there are no deer population to consume them after week 35. Before week 35, we observe everything to be almost the same as the environment without zombies. We observe that the very first zombie outbreak is at week 10, then 25, then 35. Other than that, deer reproduce at a random rate as food(grain) is always available.



Units:

- NowPrecip
  - cm
- NowTemp
  - Celsius
- NowHeight
  - cm
- NowNumDeer
  - count

Without the agent, we observe that we are able to obtain the same chart as the example, telling us that our implementation of the environment is correct.

Conclusion

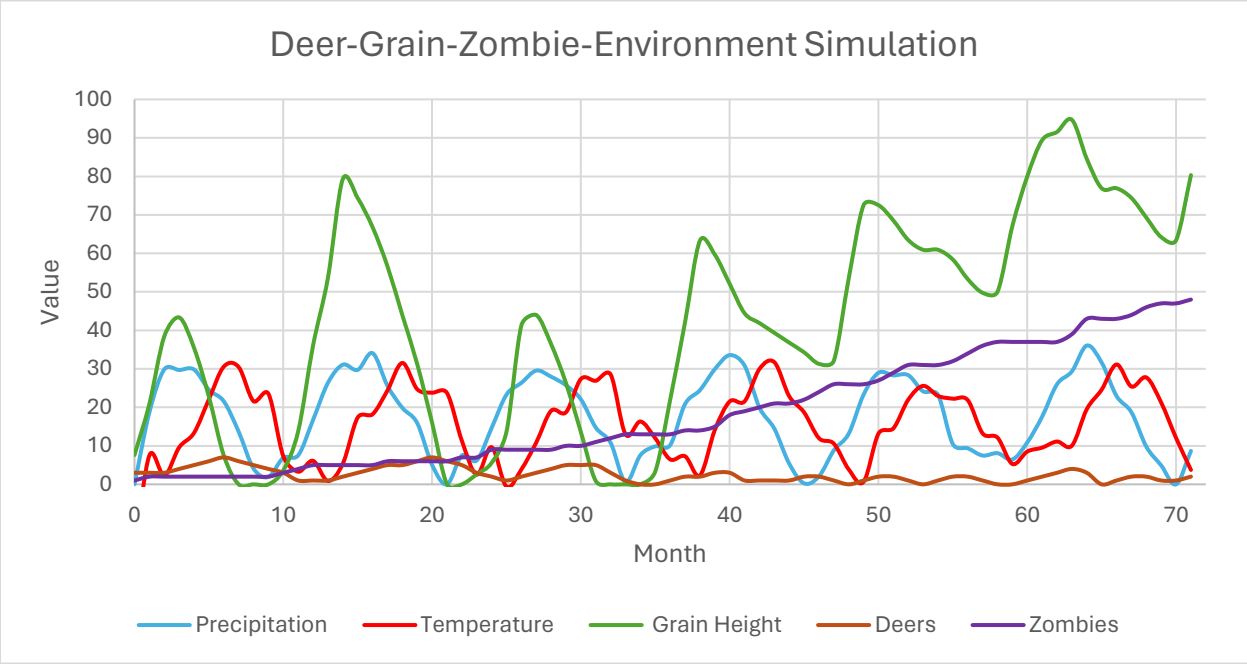
1. What your own-choice quantity was and how it fits into the simulation.

The quantity I chose is zombies, they have a chance to infect a deer and possess them, allowing them to infect others, the chance of infection is set at 0.05, or 5%. Meaning, each zombie has 5% chance to infect a deer every month, if there are more deer, there is a chance for all of them to be infected and turned into zombies. These zombies are not counted as deer and is removed from the deer population.

2. A table showing values for temperature, precipitation, number of deer, height of the grain, and your own-choice quantity as a function of month number.

Shown above in **Tables of data** section.

3. A graph showing temperature, precipitation, number of deer, height of the grain, and your own-choice quantity as a function of month number.



Units:

- Precipitation
  - cm
- Temperature
  - Celsius
- Grain Height
  - cm
- Deers
  - count
- Zombies
  - count

4. A commentary about the patterns in the graph and why they turned out that way. What evidence in the curves proves that your own quantity is actually affecting the simulation correctly?

We observe that the added zombies started out at 1, with an infection rate of 0.05, and with zombies being unable to die, the population never goes down this will lead to deer being unable to survive due to having too many zombies. Given that there are enough food in week 35+, we should be able to see deer population grow, but such is not possible because of zombies. When deer population grows, it is also easier for zombies to infect deer as there are more of them.

Comparing with observation of the chart without agent and the predictions of grain/deer behavior due to zombies, we can safely conclude that the implementation is correct. Evidence includes, deer population

being unable to grow even when there is food if there are a lot of zombies; in the initial stages, zombies are scarce, it is unable to infect deer and we would observe no change in the environment; etc.... The change of deer will affect grains as there would not be deer to eat the grains, this is also reflected in our graph.

How did things turn out?

Things turned out well with no issues. The only thing that caught me by surprise is the increase in grain due to a rise in zombies.

Why do you think they turned out that way?

I did not think that grains would go up when zombies increase due to a rollercoaster of events caused by zombies which affects deer which affects grains. Once I saw that the grains went up, it was obvious that it is not erroneous, and I was able to conclude that it made sense due to the nature of things.