ECE 277, FALL 2020
GPU Programming

Department of Electrical and Computer Engineering
University of California, San Diego

LAB 1: Setup reinforcement learning environment

This lab requires to design an agent to interact with the RL environment in Figure 1. In this lab, there is no reward from environment for simplicity. Specifically, you need to move a single thread agent to arrive the flag safely. In every step, your agent need to take an action to arrive the flag for a given current state S.

No reinforcement learning is required in this lab. This lab is just a warm-up lab to learn how to interactive with RL-environment using CUDA. You need to just move the agent to arrive the flag by your self (no RL algorithm).

The agent should interact with the given mine game environment in Figure 2.

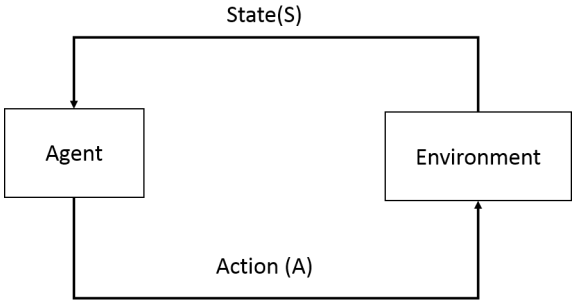

Figure 1: Reinforcement learning environment.

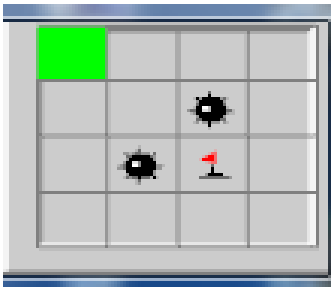

Figure 2: 4x4 mine game environment.

- Action: (right:0, down:1, left:2, up:3)
- State: (x, y) current position of an agent in the coordinator of (0,0) at the top-left corner

- Every episode restarts from (0,0) after the agent reaches one of mines or a flag.

You should not modify any given codes except CMakeLists to add your codes under "Qagent" directory. You only need to add your agent code to the lab project.

You have to use CUDA to program a single agent with

`Agent_action<<<1, 1>>>, Agent_init<<<1, 1>>>.`

Interface pointers of all the extern functions are allocated to the Device (GPU) memory (not CPU memory)

The below function is an informative RL environment routine to show when and how agent functions are called

```
extern void agent_init();
extern short* agent_action(int2* cstate);
int qlearningCls::learning(int *board, unsigned int &episode, unsigned int &steps)
    if (m_episode == 0 && m_steps==0) {// only for first episode
        env.reset(m_sid);
        agent_init(); // agent self initialization (allocate necessary memory)
    }else {
        active_agent = checkstatus(board, env.m_state, flag_agent);

        if (m_newepisode) {
            env.reset(m_sid);
            m_steps = 0;
            m_episode++;
        }else {
            short* action = agent_action(env.d_state[m_sid]);
            env.step(m_sid, action);

            m_sid ^= 1;
            episode = m_episode;
            steps = m_steps;
        }}
    m_steps++;
    env.render(board, m_sid);
    return m_newepisode;
```

Here is a list you should do for lab 1.

1. agent_init() and short* agent_action(int2* cstate) are interface functions for RL environment and they are CPU functions.
2. The agent_init() function should allocate a short-type global memory for an action buffer ("d_action")
3. Create short* agent_action(int2* cstate) in CPU
4. The agent_action function invokes an CUDA kernel (e.g. cuda_agent) for an agent action

5. cuda_agent derives cstate[0].x and cstate[0].y to identify a current agent position within an environment.

   agent_action CPU function cannot directly access the cstate pointer since it is allocated in the GPU.

6. cuda_agent takes an action to move the agent based on the current position (cstate[0].x, cstate[0].y) to arrive the flag safely.

   You can only move one position at each step

7. cuda_agent updates the action buffer with the new action, "d_action[0] = action;"

8. agent_action return the action buffer pointer to RL environment (already given).

9. submit only agent.cu file into the assignment.

Programming language: CUDA