```cpp
#include <fstream>
#include <string>
#include <iostream>

#include "engine.h"
#include "constants.h"

using namespace std;

void loadAnimationMap(string gameFile, int& numStates, int* & animationSize, int** &
animationMap)
{
    //Declare and open filestream
    fstream fin;
    fin.open(gameFile.c_str(),ios::in);

    //Number of columns to store
    fin >> numStates;

    //Allocate memory
    animationSize = new int[numStates];
    animationMap = new int*[numStates];

    //Load the ragged array
    for(int i=0;i<numStates;i++)
    {

        fin >> animationSize[i];
        animationMap[i] = new int[animationSize[i]];

        for(int j=0;j<animationSize[i];j++)
        {
            fin >> animationMap[i][j];
        }
    }

    //Clean-up
    fin.close();
}

void initLinkData(int & linkPosX, int & linkPosY, LK_STATE & linkState, int & linkAnimID, int**
animationMap, int & linkSpriteID)
{
    //Pick a random center x location for the sprite.
    linkPosX = 0;

    //Figure out what the highest height below the sprite would be.
    linkPosY = SCREEN_HEIGHT-LK_SPRITE_HEIGHT;

    //Intialize State
    linkState = STILL_RIGHT;

    //Initialize animationID
    linkAnimID = 0;
```

```
    //Pick the stationary facing left sprite
    linkSpriteID = animationMap[linkState][linkAnimID];

}


void changeGameState(LK_TRANSITION command, int & linkPosX, int & linkPosY, LK_STATE & linkState
, int & linkAnimID, int* animationSize, int** animationMap, int & linkSpriteID)
{
    switch(command)
    {
        //Execute the appropriate state transition
    case ATTACK: moveAttack(linkState,linkAnimID,animationSize, animationMap,linkSpriteID,
    linkPosX,linkPosY); break;
    case DOWN: moveDown(linkState,linkAnimID,animationSize, animationMap,linkSpriteID,linkPosX,
    linkPosY); break;
    case LEFT: moveLeft(linkState,linkAnimID,animationSize, animationMap,linkSpriteID,linkPosX,
    linkPosY); break;
    case RIGHT: moveRight(linkState,linkAnimID,animationSize, animationMap,linkSpriteID,linkPosX
    ,linkPosY); break;
    case NA: noAction(linkState,linkAnimID,animationSize, animationMap,linkSpriteID,linkPosX,
    linkPosY); break;
    }

    //Update the sprite
    updateSprite(linkState,linkAnimID,animationSize,animationMap,linkSpriteID);

    //Boundary Checking (left and right boundaries)
    if(linkPosX >= SCREEN_WIDTH-LK_SPRITE_WIDTH)
    {
        linkPosX = SCREEN_WIDTH-LK_SPRITE_WIDTH;
    }
    if(linkPosX <= 0)
    {
        linkPosX = 0;
    }

    //Boundary Checking (top and bottom boundaries)
    if(linkPosY >= SCREEN_HEIGHT-LK_SPRITE_HEIGHT)
    {
        linkPosY = SCREEN_HEIGHT-LK_SPRITE_HEIGHT;
    }
    if(linkPosY <= 0)
    {
        linkPosY = 0;
    }
}

void moveAttack(LK_STATE & linkState, int & linkAnimID, int* animationSize, int** animationMap,
int & linkSpriteID, int & linkPosX, int & linkPosY)
{

    switch (linkState)
```

```
    {
    case STILL_RIGHT:
        linkState = ATTACK_RIGHT;
        linkAnimID = 0;
        break;
    case STILL_LEFT:
        linkState = ATTACK_LEFT;
        linkAnimID = 0;
        break;
    default:
        break;
    }

}

void moveDown(LK_STATE & linkState, int & linkAnimID, int* animationSize, int** animationMap,
int & linkSpriteID, int & linkPosX, int & linkPosY)
{
    //Changed states: initialize this state
    if(linkState==STILL_LEFT)
    {
        linkState = CROUCH_LEFT;
        linkAnimID = 0;
    }
    else if(linkState==STILL_RIGHT)
    {
        linkState = CROUCH_RIGHT;
        linkAnimID = 0;
    }



}

void moveRight(LK_STATE & linkState, int & linkAnimID, int* animationSize, int** animationMap,
int & linkSpriteID, int & linkPosX, int & linkPosY)
{

    //Conduct the appropriate state transition and/or animation
    switch(linkState)
    {
    case STILL_RIGHT:
        linkState = WALK_RIGHT;
        linkAnimID = 0;
        linkPosX += LK_WALK_SPEED;
        break;
    case STILL_LEFT:
        linkState = STILL_RIGHT;
        linkAnimID = 0;
        break;
    case WALK_RIGHT:

        linkPosX += LK_WALK_SPEED;
        break;
```

```
    default:
        linkState=STILL_RIGHT;
        linkAnimID = 0;
    }

}


void moveLeft(LK_STATE & linkState, int & linkAnimID, int* animationSize, int** animationMap,
int & linkSpriteID, int & linkPosX, int & linkPosY)
{
    //Conduct the appropriate state transition and/or animation
    switch(linkState)
    {
    case STILL_LEFT:
        linkState = WALK_LEFT;
        linkAnimID = 0;
        linkPosX -= LK_WALK_SPEED;
        break;
    case STILL_RIGHT:
        linkState = STILL_LEFT;
        linkAnimID = 0;
        break;
    case WALK_LEFT:
        linkPosX -= LK_WALK_SPEED;
        break;
    default:
        linkState=STILL_LEFT;
        linkAnimID = 0;
    }

}

void noAction(LK_STATE & linkState, int & linkAnimID, int* animationSize, int** animationMap,
int & linkSpriteID, int & linkPosX, int & linkPosY)
{

    if(linkState!=STILL_LEFT || linkState!=STILL_RIGHT)
    {
        //Conduct the appropriate state transition
        switch(linkState){
        case ATTACK_RIGHT:
        case CROUCH_RIGHT:
        case WALK_RIGHT:
            linkState=STILL_RIGHT;
            break;
        case ATTACK_LEFT:
        case CROUCH_LEFT:
        case WALK_LEFT:
            linkState=STILL_LEFT;
            break;
        }
        //Reset animation and update the sprite
        linkAnimID=0;
```

```
    }
}

void updateSprite(LK_STATE linkState, int & linkAnimID, int* animationSize, int** animationMap,
int & linkSpriteID)
{
    linkAnimID++;

    //Wrap animation sequence
    if(linkAnimID>=animationSize[linkState])
    {
        linkAnimID = 0;
    }

    //Map sprite ID
    linkSpriteID = animationMap[linkState][linkAnimID];
}
```