```cpp
//System Headers
#include <fstream>
#include <cstdlib>
#include <ctime>
#include <string>
#include <iostream>

//GUI Headers
#include "GUI.h"
#include "Timer.h"

//Game Headers
#include "Engine.h"
#include "constants.h"

using namespace std;

int main(int argc, char *argv[]){

    //Seed random number generator
    srand((unsigned)time(0));

    //Initialize GUI->Engine communication
    LK_TRANSITION command = NA;

    //Construct Timer
    Timer fps;

    //Construct GUI
    GUI gui;

    //Construct Engine
    Engine engine;

    //Initialize Termination Criteria
    bool quit = false;

    //While the user hasn't quit
    while(quit == false){

        //Start the frame timer
        fps.start();

        //GUI waits for mouse events
        while(gui.observeEvent()){

            //Receive Command From GUI
            command = gui.getCommand();

            //GUI transmits quit event
            if(gui.quitGame()){
                quit=true;

            }
        }

        //Update Game state
        engine.changeGameState(command);

        //Render Game Data
        gui.displayGameState(engine);

        //Adjust Frame Rate
        if(fps.get_ticks() < FRAME_DELAY){
            SDL_Delay( FRAME_DELAY - fps.get_ticks() );
        }

    }

    //Return
    return 0;

}
```

```
#ifndef ENGINE_H
#define ENGINE_H

//System Headers
#include <string>

//Game Headers
#include "Object.h"

using namespace std;

class Engine
{

public:

    //Constructor
    Engine();

    //Accessor Methods
    Object* getObject(int);
    int getNumObjects();
    bool getGameOver();

    //Game specific method
    void changeGameState(LK_TRANSITION);


private:

    int numObjects;
    Object** objects;
    bool gameOver;

};

#endif
```

```cpp
//System Headers
#include <fstream>
#include <string>
#include <iostream>

//Game Headers
#include "constants.h"
#include "Engine.h"
#include "Object.h"
#include "Link.h"
#include "Block.h"
#include "Deeler.h"

using namespace std;

Engine::Engine()
{

    numObjects = 0;
    objects = NULL;

    //Open configuration file
    fstream fin;
    fin.open("./Assets/Config/level.txt",ios::in);

    //Get Number of Objects in configuration file
    fin >> numObjects;

    //Construct an array to store the game's objects
    objects = new Object*[numObjects];

    //Convert the configuration file into game objects
    int id = -1, x = -1, y = -1;

    //Each line of the config file specifies an object
    for(int i=0;i<numObjects;i++)
    {
        //Read in the data
        fin >> id >> x >> y;

        //Construct the appropriate object
        switch(id)
        {
        case LINK: objects[i] = new Link(x,y); break;
        case BLOCK: objects[i] = new Block(x,y); break;
        case DEELER: objects[i] = new Deeler(x,y); break;
        }

    }

    //Clean-up
    fin.close();

    //Set end-of-game condition
    gameOver = false;

}

Object* Engine::getObject(int index)
{
    Object* result = NULL;
    if(index>=0 && index<numObjects)
    {
        result = objects[index];
    }
    return(result);
}

int Engine::getNumObjects()
{
    return(numObjects);
}

bool Engine::getGameOver()
{
    return(gameOver);
}

void Engine::changeGameState(LK_TRANSITION command)
{
    if(!gameOver)
    {
        //------------------
        //Update all objects
        //------------------
        for(int i=0;i<numObjects;i++)
        {
            objects[i]->update(command);
        }

        //--------------------
        // Implement Scrolling
        //--------------------

        //Find the Link pointer
        Object* link = NULL;
        for(int i=0;i<getNumObjects();i++)
        {
            Object* object = getObject(i);
            if(object->getObjectID()==LINK)
            {
                link = object;
            }
        }

        //Scroll the game objects (compare x-position to Link)
        if(link->getPosX() > SCREEN_WIDTH/2)
        {
            link->setPosX(SCREEN_WIDTH/2);

            //Scroll each object individually
            for(int i=0;i<numObjects;i++)
            {
                Object* object = getObject(i);
                if(object->getObjectID() == BLOCK || object->getObjectID() == DEELER)
                {
                    object->setPosX(object->getPosX()-LK_RUN_SPEED);
                }
            }
        }

        //--------------------
        // Detect end-of-game
        //--------------------

        //Find Deeler and detect collision with Link
        for(int i=0;i<numObjects;i++)
        {
            Object* object = getObject(i);
            if(object->getObjectID() == DEELER)
            {
                //Find center of link
                int linkLeftX = link->plotX();
                int linkTopY = link->plotY();
                int linkRightX = link->plotX() + LK_SPRITE_WIDTH;
                int linkBottomY = link->plotY() + LK_SPRITE_HEIGHT;

                int deelerLeftX = object->plotX();
                int deelerTopY = object->plotY();
                int deelerRightX = object->plotX() + DEELER_SPRITE_WIDTH;
                int deelerBottomY = object->plotY() + DEELER_SPRITE_HEIGHT;


                //Detect link collision with Deeler
                if (
                    ((linkRightX > deelerLeftX &&
                    linkRightX < deelerRightX) ||
                    (linkLeftX < deelerRightX &&
                        linkLeftX > deelerLeftX))
                    &&
                    linkTopY < deelerBottomY
                    )
                {
                    gameOver = true;
                }
            }
        }
    }
}
```

```cpp
#ifndef OBJECT_H
#define OBJECT_H

#include "constants.h"
using namespace std;

class Object
{

public:

    Object(int,int);

    //Get Methods
    float getPosX() const;
    float getPosY() const;
    int getObjectID() const;
    int getSpriteID() const;

    //Set Methods
    void setPosX(float);
    void setPosY(float);
    void setObjectID(OBJECT_ID);
    void setSpriteID(int);

    //Plotting Methods
    int plotX();
    int plotY();

    //Animation Function
    virtual void update(LK_TRANSITION)=0;

protected:

    //Declare Object Properties
    float posX;
    float posY;
    OBJECT_ID objectID;
    int spriteID;

};

#endif
```

```cpp
#include "constants.h"
#include "Object.h"

using namespace std;

Object::Object(int posX, int posY){

    //Convert to floating point internal representation
    this->posX = (float)posX;
    this->posY = (float)posY;


    //Assign base object type
    objectID = OBJECT;

    //No base art asset
    spriteID = -1;
}

//Get Methods
float Object::getPosX() const{
    return(posX);
}

float Object::getPosY() const{
    return(posY);
}

int Object::getObjectID() const{
    return(objectID);
}


int Object::getSpriteID() const{
    return(spriteID);
}

//Set Methods
void Object::setPosX(float posX){
    this->posX = posX;
}

void Object::setPosY(float posY){
    this->posY = posY;
}


void Object::setObjectID(OBJECT_ID objectID){
    this->objectID = objectID;
}

void Object::setSpriteID(int spriteID){
    this->spriteID = spriteID;
}
```

```cpp
//Plotting Methods
int Object::plotX(){
    return((int)posX);
}

int Object::plotY(){
    return((int)posY);
}
```

```cpp
#ifndef Link_H
#define Link_H

//System Headers
#include <string>

//Game Headers
#include "Object.h"
#include "constants.h"

using namespace std;

class Link: public Object
{

public:

    Link(int,int);
    ~Link();

    //Get Methods
    float getVelY() const;
    int getState() const;

    //Set Methods
    void setVelY(float);
    void setState(int);

    //Animation Function
    void update(LK_TRANSITION);

protected:

    //Declare class physics properties
    float velY;

    //Declare class specific properties
    int state;
    int animationID;

    //Animation Storage
    int numStates;
    int* animationSize;
    int** animationMap;

    //Private functions to manipulate internal class state
    void loadAnimation(string);
    void updateSprite();
    void moveAttack();
    void moveDown();
    void moveLeft();
    void moveRight();
    void noAction();
};

#endif
```

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

//Game Architecture Headers
#include "constants.h"
#include "Link.h"

Link::Link(int posX, int posY): Object(posX,posY){

    //Identify the object type
    objectID = LINK;

    //Load Animation Data
    loadAnimation("./Assets/Config/animation.txt");

    //Initialize the Link's game/animation state
    state = STILL_RIGHT;
    animationID = 0;

    //Initialize physics
    velY = 0.0f;

    //Compute Initial SpriteID
    updateSprite();

}


Link::~Link(){

    //Clean-up ragged 2D array
    for(int i=0;i<numStates;i++){
        delete [] animationMap[i];
    }
    delete [] animationMap;

    //Clean-up 1D array
    delete [] animationSize;

}

void Link::loadAnimation(string gameFile){

    //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();

}

//Get Methods
int Link::getState() const{
    return(state);
}

float Link::getVelY() const{
    return(velY);
}


//Set Methods
void Link::setState(int state){
    this->state = state;
}

void Link::setVelY(float velY){
    this->velY = velY;
}

void Link::update(LK_TRANSITION command){

    switch (command) {
        //Execute the appropriate state transition
    case ATTACK: moveAttack(); break;
    case DOWN: moveDown(); break;
    case LEFT: moveLeft(); break;
    case RIGHT: moveRight(); break;
    case NA: noAction(); break;
    }

    //Apply Physics
    posY += velY;
    velY += DELTA_T*GRAVITY;

    //Left Boundary Detect & Resolve
    if(plotX() <= 0){
        setPosX(0);
    }

    //Bottom Boundary Detect & Resolve
    if(getPosY() >= SCREEN_HEIGHT-LK_SPRITE_HEIGHT){
        setPosY((float)(SCREEN_HEIGHT-LK_SPRITE_HEIGHT));
        velY = 0.0f;  //Stops falling
    }

    //Top Boundary Detect & Resolve
    if(getPosY() <= 0){
        setPosY(0.0f);
    }
}

void Link::moveAttack(){

    //Changed states: initialize this state
    switch (state)
    {
    case STILL_RIGHT:
        state = ATTACK_RIGHT;
        animationID = 0;
        break;
    case STILL_LEFT:
        state = ATTACK_LEFT;
        animationID = 0;
        break;
    default:
        break;
    }

    updateSprite();
}

void Link::moveDown(){

    switch (state)
    {
    case STILL_LEFT:
        state = CROUCH_LEFT;
        animationID = 0;
        break;
    case STILL_RIGHT:
        state = CROUCH_RIGHT;
        animationID = 0;
        break;
    }
    updateSprite();

}

void Link::moveRight() {

    //Conduct the appropriate state transition and/or animation
    switch (state) {
    case STILL_RIGHT:
        state = WALK_RIGHT;
        animationID = 0;
        posX += LK_RUN_SPEED;
        break;
    case WALK_RIGHT:
        posX += LK_RUN_SPEED;
        break;
    default:
        state = STILL_RIGHT;
        animationID = 0;
    }
    updateSprite();
}


void Link::moveLeft() {

    //Conduct the appropriate state transition and/or animation
    switch (state) {
    case STILL_LEFT:
        state = WALK_LEFT;
        animationID = 0;

        posX -= LK_RUN_SPEED;
        break;
    case WALK_LEFT:

        posX -= LK_RUN_SPEED;
        break;
    default:
        state = STILL_LEFT;
        animationID = 0;
    }
    updateSprite();

}

void Link::noAction() {

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

}

void Link::updateSprite() {

    animationID++;

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

    //Map sprite ID
    spriteID = animationMap[state][animationID];
}
```

```cpp
#ifndef BLOCK_H
#define BLOCK_H

//System Headers
#include <string>

//Game Headers
#include "Object.h"

using namespace std;

class Block: public Object
{

public:

    Block(int,int);

    //Action methods
    void update(LK_TRANSITION);

protected:


};

#endif
```

```cpp
#include "constants.h"
#include "Block.h"

using namespace std;

Block::Block(int posX, int posY):Object(posX,posY)
{
    objectID = BLOCK;
}

void Block::update(LK_TRANSITION command)
{
    //Do nothing
}
```

```cpp
#ifndef DEELER_H
#define DEELER_H

//System Headers
#include <string>

//Game Headers
#include "Object.h"

using namespace std;

class Deeler: public Object
{

public:

    Deeler(int,int);

    //Action methods
    void update(LK_TRANSITION);

protected:

    bool rise;
    int moveCount;
    int moveCountMax;

};

#endif
```

```cpp
#include "constants.h"
#include "Deeler.h"

using namespace std;

Deeler::Deeler(int posX, int posY):Object(posX,posY)
{
    objectID = DEELER;
    rise = true;
    moveCount = 0;
    moveCountMax = 50;
}

void Deeler::update(LK_TRANSITION command)
{

    if(rise)
    {
        if(moveCount<moveCountMax)
        {
            posY-=DEELER_SPEED;
            moveCount++;
        }
        else
        {
            rise = false;
            moveCount = 0;
        }
    }
    else
    {
        if(moveCount<moveCountMax)
        {
            posY+=DEELER_SPEED;
            moveCount++;
        }
        else
        {
            rise = true;
            moveCount = 0;
        }
    }
}
```