# Chien 2D: A Multiplatform Library to Teach the C Language Through Games Programming

Paulo V. W. Radtke
Setor de Educação Profissional e Tecnológica
Universidade Federal do Paraná

Fabio V. Binder
Pontifícia Universidade Católica do Paraná

## Abstract

This paper introduces *Chien 2D*, a C game programming library that target problem-based learning of the C language. This approach is interesting to raise interest on students and reduce the student dropout rate. We have considered the limitations of a begginer's C programmer, allowing appealing results with very simple programming structures, and yet providing enough features for more complex development. The library has been successfully used with the *Games Tutorial* methodology, improving students programming performance.

**Keywords::** C language, games programming, 2D games, multiplatform

**Author's Contact:**

paulo.radtke@ufpr.br
fabio.binder@pucpr.br

## 1 Introduction

From internet banking to social networks, most of our daily activities are assisted by computer programs. Computer programmers have become a common presence in the most diverse environments, helping to adapt and improve daily tasks on information systems. However, several IT industry reports have observed that the number of open positions increases year after year [Jr. 2009]. It is also observed that undergrad computer related courses in Brazil are affected by a high student drop out rate [Takahashi 2009], and one reason is pointed most often than others: the difficulty to learn programming [Kinnunen and Malmi 2006].

Programming lecturers usually challenge students with synthetic problems to exercise programming concepts. This approach usually fails to relate programming structures to solve more complex problems, owing to limited exercise scope and laboratory time. Instead, students can improve their skills when challenged to solve more complex problems that they can relate to. The problem-based learning [Kinnunen and Malmi 2005] approach on programming languages has students working on larger problems, where multiple concepts are applied together. In this sense, we have used game programming as a problem to help students to learn program the C language. We have observed a similar approach presented in [Bayliss and Strout 2006], which encourages the adoption of activities that engage the students on subjects that they can relate to.

Our approach is divided in two main aspects, a *Games Tutorial* methodology to steer the learning process and a programming library to provide a simple and powerful framework to develop games. The *Games Tutorial* encourages teamwork on a long term project, requiring students to identify issues and apply knowledge unrelated to the programming course, but to the problem (game) they are developing [Stroustrup 2010]. The *Chien 2D* library, detailed in Section 2, is a multiplatform C game programming library, focused on 2D graphics, which allows students to easily develop games on C. An example to demonstrate the library is discussed in Section 4.

This combined approach has been applied at the Pontifícia Universidade Católica do Paraná (PUCPR) since 2006, and at the Federal University of Paraná (UFPR) starting 2010. The main contributions of this work are threefold. We first observed that students that took part of this activity improved their programming and teamwork skill when compared to students that relied only on the tradicional approach. Next, there was a positive impact on the student drop-out rate, as indicated by numbers prior and after the *Games Tutorial* started. Both contributions are further discussed in Section 5. Finally, we provide a simple and yet powerful programming library, which is open source under the Apache 2.0 license.

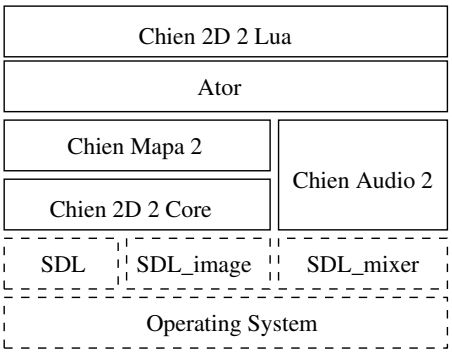


**Figure 1:** *Chien 2D library architecture.*

## 2 Chien 2D Library

Using game programming on a problem-based approach to teach the C programming language presented us some challenges to overcome. When students start learning C, their knowledge is limited to the use of built in data types and simple conditional and loop programming structures. Later on, they gain knowledge to use arrays, structure their programs with modular functions and to represent heterogeneous information through data structures. Finally, after mastering those concepts, students start working with pointers, dynamic memory allocation and files. The goal is to provide a framework that students can use to develop very simple games at the beginning, but with enough advanced features to incrementaly exercise more complex concepts unrelated to programming. Simply using a traditional game programming library is not adequate in this context. Libraries such as SDL [Latinga 1998] and Allegro [Hargreaves 1995] demand at least intermediate C language knowledge to perform the most trivial tasks.

The *Chien 2D* library achieves those requirements, and yet it provides enough advanced features to allow students to develop full games after mastering the C programming language. A modular design, depicted in Fig. 1 is used to divide several features into a series of smaller libraries, which can be combined as required by specific projects. The library currently wraps around the SDL library, which provides actual access to system resources. Whereas games have to be linked with the SDL library binaries, no knowledge of these libraries or the underlying operating system is required. Thus, dashed modules in Fig. 1 are indicated for completeness. After an initial development, the library reached version 2 in 2007, which is stable, and maintenance is dedicated to add new features or improve performance.

The library is freely available for download at **http://code.google.com/p/chien2d/**, ready for compiling under GNU Linux and Windows. Whereas no official support is provided, we have reports of students that succesfully used the library on XCode (Mac OS X). Students start out by learning the *Chien 2D* core library, moving on to other optional libraries as both the C language knowledge improves and game complexity increases. The following list detail the modular libraries.

- **Chien 2D Core**: the core library is sprite based, allowing programmers to load images and split them up in several animation frames, with either color key (magenta) or alpha channel transparency. The library provides both software and OpenGL rendering for faster operations and some extra effects. Text output is supported using bitmapped fonts generated by Bitmap Font Builder [Wetzel 2000], and basic user input is provided. Functions are simple and manipulate only integer values, avoiding pointer usage at early stages.
- **Chien Audio**: this module adds audio support to games, using two different streams, one for music and other for sound effects, that are mixed on a simple stereo output. Most audio formats are supported, being the most often used the Microsoft Wave, Ogg Vorbis and MP3. Only one music can be played at a time, but up to 8 sound effects may be mixed at once. At this point, the lecturer may introduce to students relevant issues on software licensing. Among the supported audio formats, MP3 playback requires a paid license for commercial software, whereas other formats, such as Ogg Vorbis, don't.
- **Chien Mapa**: static images can be used as backgrounds for simple games. But as complexity scalates and playfield movement is required, one screen may not provide enough room. Tilemap support allows large playfields with efficient memory usage. This module uses tilemaps generated by Mappy [Burrows 2005], supporting animated tilemaps and tags to describe each block. The library separates the tileset (sprites) from the actual tilemap, to allow the usage of the same tileset on several tilemaps to reduce disk usage. Given a character bounding box, the library provides basic functions to test collisions against special tagged blocks and to avoid movements over walls.
- **Ator**: game characters on action games interact with both the environment and other characters. This module provides support to develop characters for sidescrolling and top view action games, including gravity simulation. Character movement is restricted by the environment, and when jumping on sidescrolling games, characters will automatically fall until they hit the floor. This module fully exercises the usage of pointers, including a pointer to a function that implements the actual character behavior. Also, the module introduces finite state automata to students, using a state to describe the characters current situation and events to change the state accordingly.
- **Chien 2D Lua**: the final module binds the C language libraries to the Lua scripting language [Ierusalimschy et al. 2006]. This module is intended for advanced studies, usually in the last part of a two semester C undergrad course, or at specific game development lectures. The advantage is to provide easy manipulation of character behavior outside the programming environment, which may allow character designers to script characters without getting involved with the C programming aspects.

# 3 Games Tutorial Methodology

Traditional books used to learn C, such as [Schildt 1997], follow a similar structure when presenting the C programming language. Generally speaking, topics are presented in an order similar to the following non-exhaustive list: identifiers and data types, operators and expressions, standard input and output, conditional and iteration statements, arrays, structures and data type definition, functions and structured programming, pointers and files.

Traditional game programming APIs are not easy for students that mastered only the first four items in the previous list. However, this is exactly what *Chien 2D* achieves in C, which allowed us to formulate the *Games Tutorial* as an extension activity for first year students. The *Games Tutorial* is planned to last one year with one weekly meeting, the traditional length of a C programming lecture.

The *Games Tutorial* is divided in four stages. Each stage considers the students current programming knowledge. Thus, any difficulties faced by students will be directly related to recent lectures, and solving those difficulties will actually improve their programming knowledge. The following list details the four stages developed during the *Games Tutorial*, each with a half semester length.

- **Game project and architecture**: the first stage introduces 2D game concepts, such as sprites, tilemaps, graphics and audio file formats and animation. We also introduce a simple software development lifecycle, focused on games, from requirements to testing and documentation. A synthetic game project is discussed, and students are encouraged to choose a project and write a document detailing their design.
- **Simple game development**: the *Chien 2D* library is presented through examples and challenges to modify these examples. Section 4 details a simple baloon pop game.
- **Prototyping**: here students implement the user interface (menus) for their project.A very basic game interface is expected at this point.
- **Full game development**: the final stage has students implementing the actual game code. Students also have to provide a website for their game and a manual, detailing the installation procedure and gameplay rules. Finally, a local game fair is organized yearly, after the *Games Tutorial* activities is completed, so students can present their work to the public.

Being an activity related to the programming language lecture, it is important that both activities are synchronized. Students will feel unconfortable if topics used during the *Games Tutorial* have not yet been introduced on the regular lecture. As an optional activity, integrating it as part of the student grading on the programming lecture will help raise interest among students. This in turn tends to help students, as the extra exercise helps them consolidate and improve their programming language knowledge.

Also, the *Games Tutorial* methodology is in no way tied to the C language and *Chien 2D*. We have also used the same methodology with JavaME at both PUCPR and UFPR and results were encouraging. However, careful planning must be made in order to select a suitable programming approach to another context. Otherwise, students may become frustrated by having difficulties to learn both programming and game development.

# 4 Programming Example

To illustrate game development using only basic C and *Chien 2D* function calls, a ballon pop game is demonstrated in this section. This example is the last one on a four stages incremental example set. Each example challenges students to try out their programming skills, in order to improve the application, as adding more balloons or using different sprite colors for each balloon. The final example is the starting point for a complete casual game, where the student can implement menu navigation, high score tables and incremental game difficulty. This approach allows students to both learn the *Chien 2D* library and practice C programming. Unlike synthetic lecture examples to illustrate the C statements, the concrete results obtained through those examples encourage students to improve them and, in consequence, to better understand those statements and functions. The examples are available at the official *Chien 2D* library repository at **http://code.google.com/p/chien2d/**, under the *Exemplos/sbgames* folder. Instructions to build, install the library and compile the examples is available on README files for both GNU Linux and Windows plaftforms.

The balloon pop game used to introduce the library is detailed in Listing 1. It loads a 35x50 pixels balloon image, a 21x21 pixels mouse cursor and a 64 pixels tall font. It sets up a windowed game screen of 800x600 pixels and allows the player to pop ballons, keeping track of the current player score. To compile this example on a GNU environment with the *g++* compiler, we can use the following command:

```
g++ exemplo04.cpp -o exemplo04 -lGL -lGLU
     -lSDL -lSDL_image -lc2d2
```

The only library include used refers to the core *Chien 2D* library, *c2d2/chien2d2.h*, and defines all core functions and symbolic constants. The balloons position are stored in the *baloes* array, where each line indicates the *(x, y)* balloon's coordinate. The *conta* auxiliar variable is used in for loops, and the *score* variable is used to track the player's current score. The *mouse* and *teclas* pointer let the programmer access the computer mouse and keyboard. On *Chien 2D*, the keyboard is managed as an indexed array of buttons (**C2D2_Botao** datatype), where each button is associated to a key through a symbolic constant. To retrieve the buton array, we use a pointer to **C2D2_Botao** and associate to it the return of the **C2D2_PegaTeclas** function. Each button is a structure with three fields, *ativo*, which indicates if the user is pressing the key, *pressionado*, which indicates that the user has just pressed the key and *liberado*, indicating that the user has just released the key. Each key is indexed through symbolic contants and, on our example, are the escape key (**C2D2_ESC**) and the button to close the window (**C2D2_ENCERRA**), which is mapped to both the graphical icon on the window and to the *ALT+F4* key combination on both GNU Linux and windows. The mouse is a data structure of the **C2D2_Mouse** type, which has fields **x** and **y** for its coordinates inside the game window, and **botoes** for the mouse buttons (also using the **C2D2_Botao** data type). The mouse coordinates are used to draw the cursor inside the game loop.

**Listing 1:** *exemplo04.cpp – balloon popping.*

```
#include <c2d2/chien2d2.h>
#include <stdio.h>
#include <math.h>

#define RESX      800
#define RESY      600
#define NUM_BALOES   50

int main(int ac, char **av)
{
  unsigned int balao, mira, fonte;
  int baloes[NUM_BALOES][2], conta, placar=0;
  C2D2_Botao *teclas = C2D2_PegaTeclas();
  C2D2_Mouse *mouse = C2D2_PegaMouse();
  char texto[80];
  if(!C2D2_Inicia(RESX, RESY, C2D2_JANELA,
      C2D2_DESENHO_PADRAO, "Baloes"))
    return 1;
  C2D2_TrocaCorLimpezaTela(128, 128, 255);
  for(conta=0;conta<NUM_BALOES;conta++)
  {
    baloes[conta][0] = random()%RESX;
    baloes[conta][1] = random()%RESY;
  }
  balao = C2D2_CarregaSpriteSet("balao.png",
      35,50);
  mira = C2D2_CarregaSpriteSet("mira.png", 21, 21)
      ;
  fonte = C2D2_CarregaFonte("isabelle64.png", 64);
  if(0 == balao || 0 == mira)
  {
    C2D2_Encerra();
    return 1;
  }
  while(!teclas[C2D2_ESC].pressionado && !teclas[
      C2D2_ENCERRA].pressionado)
  {
    for(conta=0;conta<NUM_BALOES;conta++)
    {
      if(C2D2_ColidiuSprites(mira, 0, mouse->x,
          mouse->y, balao, 0, baloes[conta][0],
          baloes[conta][1]) && mouse->botoes[
          C2D2_MESQUERDO].pressionado)
      {
        placar++;
        baloes[conta][0] = random()%RESX;
        baloes[conta][1] = RESY;
              }
              else if(--baloes[conta][1] < -35)
      {
        baloes[conta][0] = random()%RESX;
        baloes[conta][1] = RESY;
      }
    }
    C2D2_LimpaTela();
    for(conta=0;conta<NUM_BALOES;conta++)
      C2D2_DesenhaSprite(balao, 0, baloes[conta
          ][0], baloes[conta][1]);
    C2D2_DesenhaSprite(mira, 0, mouse->x, mouse->y
        );
    sprintf(texto, "Placar: %i pontos", placar);
    C2D2_DesenhaTexto(fonte, 0, 0, texto,
        C2D2_TEXTO_ESQUERDA);
    C2D2_Sincroniza(C2D2_FPS_PADRAO);
  }
  C2D2_RemoveSpriteSet(balao);
  C2D2_RemoveSpriteSet(mira);
  C2D2_RemoveFonte(fonte);
  C2D2_Encerra();
  return 0;
}
```

To store the images, the unsigned integer variables **balao** and *mira* are defined. The **C2D2_Inicia** function initializes the core library, taking 5 parameters: horizontal resolution, vertical resolution, screen mode (either **C2D2_JANELA** or **C2D2_TELA_CHEIA**, drawing method (**C2D2_DESENHO_PADRAO**, a software mode, or **C2D2_DESENHO_OPENGL**, an OpenGL accelerated mode) and a window title. If the **C2D2_Inicia** is successful, it returns a true value. Otherwise it returns false, and the program should terminate. Next we choose the background color that the screen is cleared to with **C2D2_TrocaCorLimpezaTela**, which takes an RGB triplet to define the background color (black by default). To load the images (a sprite set), we use the **C2D2_CarregaSpriteSet** function, which takes three parameters: the filename and the sprite's width and height. If the frame width/height are smaller than the actual image size, the image will be split up in several indexed frames. If the image is loaded correctly and the frame size is consistent, the function returns an unique positive identifier for the sprite set. Otherwise, it returns 0 on failure. On our example, the program deinitialize the *Chien 2D* core library with **C2D2_Encerra** and quits.

To draw the current score we have to print its value to a string (**texto**) using the **sprintf** C function. The core library supports sprite based text drawing. To load a font from a prepared image, we use the **C2D2_CarregaFonte** function, which takes two parameters: the font image filename and its size (whereas fonts must use square sprites, spacing considers the actual character width). This function returns the unique positive font identifier, or 0 if it failed loading the font.

The main game loop is performed until the ESC key is pressed, or the window is closed. The game logic first detects if the ballon has been hit, testing if the left mouse button is pressed and both the cursor and the balloon sprites are overlapping. The left mouse button is accessed in the *mouse* data structure in the **botoes** array, indexed by the **C2D2_MESQUERDO** symbolic constant. As a button, we use the value of the *pressionado* field to determine that we have to verify if we actually popped a baloon. Pixel level collision detection is performed by the **C2D2_ColidiuSprites** function, which returns a true or false value wether the sprites collided or not. The function takes 8 parameters, which are actually very simple after we divide them in two groups of four values. The first four values are the values used to draw the first sprite (unique identifier, sprite index, $x$ and $y$ position), and the next four values refer the values used to draw the second sprite. Each time a baloon is popped, the **placar** variable (the current score) is incremented. If the balloon has not been hit, we move it and check if it has moved outside the screen. In both situations, the balloon is respawned and the screen's bottom.

The game loop begins the screen drawing with **C2D2_LimpaTela**, which clears the screen to the color defined by **C2D2_TrocaCorLimpezaTela** (or black by default). Next we draw the sprite using **C2D2_DesenhaSprite**, which takes four parameters: the unique spriteset identifier number, the sprite index number (0 on images with only one frame) and the coordinates on the screen where we want to draw the sprite. Finally, the backbuffer

is displayed with **C2D2_SINCRONIZA**, which takes a parameter that indicates how many frames per second the application should run (**C2D2_FPS_PADRAO** is 60fps).

When the game is over, clean up is performed with two functions. **C2D2_RemoveSpriteSet** removes from memory an individual sprite, indicated by its unique integer identifier (**balao**), and **C2D2_RemoveFonte** does the same for a font. Finally, **C2D2_Encerra** deinitializes the library. The student at this point is challenged to perform the following tasks to improve the game:

- Simulate wind.
- Move baloons at different speeds.
- Slowly increase the balloon number.
- Limit the number of ballons that the player can miss.
- Add a bird character that the player can not hit
- Add a title and game over screens.
- Use different colors for the balloons.

## 5 Academic Impact

The *Games Tutorial* methodology was first implemented in 2006 at the Computer Science course at PUCPR, and results are already perceived. Students now participate on game related events, and there is a perception that the activity is relevant to learn computer programming. Some students have even chosen the course so they could follow the *Games Tutorial*. Students are motivated to develop high quality projects in two ways: each activity is graded on the *Computer Science Introduction* course, and the best games are selected to participate at the *Mostra de Jogos* (a local game fair). The game fair presents the students to the community, and a competition takes place to select the best game, following the MDA framework [Hunicke et al. 2004].

To verify the *Games Tutorial* impact, a survey was conducted with 50 students that participated in the activity betwwen 2007 and 2009 at PUCPR. For 92% of these students, the activity was relevant, and to 52% of them, the *Games Tutorial* helped learning programming concepts (functions, pointers, program structure, and files where the most cited). The activity helped 20% of the surveyed students to master the C programming language, and 13% of them enjoyed working with graphics, and also 13% where satisfied with the resulting game. Among the students that considered the *Games Tutorial* irrelevant, 60% had no interest on games, and did not completed their game project. A more detailed analysis is presented in [Binder and Martins 2010]. Another relevant aspect is that student dropout rate decreased after the Games Tutorial methodology took place. In 2008, the computer science course at PUCPR had 115 students enrolled. In 2010, a total of 142 students were enrolled. Whereas we can not say that the proposed methodology was entirely responsible for this decrease, we are confident that among all initiatives of the computer science department, the *Games Tutorial* helped decrease the student dropout rate.

Althought we targeted the *Chien 2D* library for beginners, the library also has been successfully used with post-grad students on a game specialization course at PUCPR. When working with such advanced programmers, the library has demonstrated effective to develop complex games in very short time. In this context, the Lua binding provided by *Chien 2D Lua* made it even faster to develop games, as testing and balancing are done outside a traditional programming environment.

## 6 Discussion

We have presented in this paper the *Chien 2D* library, which was sucessfully used with the *Games Tutorial* methodology. The methodology targets game development as a problem-based approach to teach the C language programming. In this problem based learning context, the library requires beginner's C programming skills, whereas other C game development libraries require at least intermediate C programming skills. The example detailed in Section 4 demonstrates that basic programming skills can produce appealing results, with enough room for students to improve the examples. The *Games Tutorial* combined with the *Chien 2D* library had a positive impact on students, as we discussed in the previous section.

The *Chien 2D* library is also open source, under the Apache 2.0 license, so other universities and groups are free to use and collaborate at the official repository. Licensing allows the production of commercial products, meaning that independent game developers also have a simple, and yet powerful library to develop or prototype games.

## Acknowledgements

The first author would like to acknowledge professors Claudio R. V. Carvilhe and Alceu de S. Britto Jr, long time colleagues, for their invaluable help to implement the *Games Tutorial* way back in 2006 at PUCPR. Both authors also acknowledge the students engagement, and their feedback to improve the library over the years. We also acknowledge professor Vidal Martins for his commitment to the *Games tutorial*.

## References

BAYLISS, J. D., AND STROUT, S. 2006. Games as a "flavor" of cs1. In *Proceedings of the 37th SIGCSE technical symposium on Computer science education*, 500–504.

BINDER, F. V., AND MARTINS, V. 2010. Uma abordagem lúdica para a aprendizagem de programação de computadores. In *WEI 2010*.

BURROWS, R., 2005. Mappy. http://tilemap.co.uk/mappy.php.

HARGREAVES, S., 1995. Allegro – a game programming library. http://alleg.sourceforge.net/.

HUNICKE, R., LEBLANC, M., AND ZUBEK, R., 2004. Mda: A formal approach to game design and game research. http://cs.northwestern.edu/ hunicke/pubs/MDA.pdf.

IERUSALIMSCHY, R., DE FIGUEIREDO, L. H., AND CELES, W. 2006. *Lua 5.1 Reference Manual*. Lua.org, August.

JR., L. C., 2009. Setor de ti aponta falta de mão-de-obra qualificada. Gazeta do Povo, 13 de Maio. http://bit.ly/bQnsOg.

KINNUNEN, P., AND MALMI, L. 2005. Problems in problem-based learning – experiences, analysis and lessons learned on an introductory ptogramming course. *Informatics in Education 4*, 2, 193–214.

KINNUNEN, P., AND MALMI, L. 2006. Why students drop out cs1 course? In *ICER '06: Proceedings of the second international workshop on Computing education research*, ACM, New York, NY, USA, 97–108.

LATINGA, S., 1998. Sdl – simple directmedia layer. http://www.libsdl.org/.

SCHILDT, H. 1997. *C: Completo e Total*. Makron Books.

STROUSTRUP, B. 2010. Viewpoint what should we teach new software developers? why? *Commun. ACM 53*, 1, 40–42.

TAKAHASHI, F., 2009. Matemática e ciências da computação têm alta taxa de abandono. Folha de So Paulo, 6 de Abril. http://bit.ly/daPZUm.

WETZEL, T., 2000. Bitmap font builder. http://www.lmnopc.com/bitmapfontbuilder/.