

A comparative approach for running a Cocos2d-x C++ cross-platform application on web browsers

Amit Bose

Master of Science in Information Technology

Kiel University of Applied Sciences

September 6, 2016

Contents

List of Figures	iv
Dedication	v
Acknowledgements	vi
Abstract	vii
1 Introduction	1
2 Application overview	2
3 Background and related work	4
3.1 Emscription	4
3.2 Google Native Client	7
3.3 Cocos	10
3.3.1 Cocos2d C++	10
3.3.2 Cocos2d-JS	11
3.3.3 Cocos2d-HTML5	12
3.3.4 Cocos2d JavaScript binding	12
3.4 FlasCC	13
3.5 CocosSharp	15
3.6 Cheerp	15
3.7 Compiler and cross platform compiler	15
3.7.1 Lexical analysis	16
3.7.2 Syntactic Analysis	17
3.7.3 Intermediate code generation	20
3.7.4 Machine code generation	21
3.7.5 Analysis and optimize	21

3.7.6	Memory management	22
3.8	Cross compiler	24
3.9	Basic difference between the structure of C++ and JavaScript code	24
4	Evaluation of approcahes	26
4.1	First Approach using Cocos sharp	26
4.1.1	setup	26
4.1.2	Approach details	26
4.1.3	Results	27
4.1.4	challenges	27
4.1.5	pros/cons	27
4.2	Second Approach using Cheerp	27
4.2.1	setup	27
4.2.2	Approach details	27
4.2.3	Results	28
4.2.4	challenges	29
4.2.5	pros/cons	29
4.3	Third approach using Google native client	29
4.3.1	setup	29
4.3.2	Approach details	30
4.3.3	Results	30
4.3.4	challenges	31
4.3.5	pros/cons	31
4.4	Fourth approach using cocos2d-js	31
4.4.1	setup	31
4.4.2	Approach details	31
4.4.3	Results	32
4.4.4	challenges	32

4.4.5	pros/cons	33
4.5	Fifth Approach using emscription	33
4.5.1	setup	33
4.5.2	Approach details	33
4.5.3	Results	35
4.5.4	challenges	35
4.5.5	pros/cons	36
5	Validation of the approaches	36
5.1	Comparison of the approach	36
5.2	Evaluation of comparison result	41
5.3	Evaluating feasibility of code for transpiling	42
6	Process of Prototyping	43
6.1	Introduction of Transpiler	43
6.2	Observation of the code structure of cocos2d C++ and cocos2d JS	48
6.3	Architecture of the transpiler	49
6.4	Method and algorithm used	51
6.4.1	Hidden markov model	51
6.5	Implementation of the transpiler	53
7	Conclusion	59
7.1	Final result	59
7.2	Limitation and deviation from hypothesis	61
7.3	Future work	61
8	Appendix	62
	References	66

List of Figures

1	TrunApp	3
2	TrunApp	4
3	NaCL toolchain and component interaction overview [22]	9
4	cocos hierarcial evolution for version 2 [14]	13
5	cocos hierarcial evolution for version 3 [15]	14
6	cocos2d-HTML5 running on web server	32
7	Summary of comparison of approaches	42
8	Sequence diagram for transpiler	56
9	Decomposed sequence diagram for transpiler for lexical	57
10	Decomposed sequence diagram for transpiler for grammar	58
11	Transcompiled code from cocos2d C++ to cocs2d JS	59
12	screenshot of the menu driven transpiler console	59
13	Screenshot of the coco2d JS application on web browser after transpiling	60

Dedication

I would like to dedicated my dissertation to my family for all the support.

Acknowledgements

I would like to thank Mario Förster senior software developer of TrunCAD GmbH for the guidance, supervision and advice for the completion of this thesis.

I would like to express my gratitude to Prof.Dr. Robert Manzke for the review and his expert guidance

I would also like to thank Prof.Dr. Jens Lüssem for the review and guidance.

Abstract

Cross platform development framework are extensively used in the organization for cost saving and easy maintenance. In this thesis i will compare the different approaches to transfer the cross-platform cocos2d C++ application into web without recoding the whole application or writing a native code for it. Here i will also discuss the different approaches that i applied for achieving my goal along with the comparative analysis and the reason for using the particular framework.

1 Introduction

For a product to really become popular it must be reachable to lot of people, this can only be done making it available to public both in the form of native app and web app. Both the form of app has its own pros and cons for example web app features like nothing to download and install make it easier for the people to it started right away removing downloading step in the process. Even company can be sure that all its clients are working on the same version in case of web app as updates are done in the server which are reflected in the browser, even from the point of business analytics also it good as it helps to give information of how user interact with the app. whereas if one is offline then native app really becomes handy. Truncad GmbH already have native app in form of TrunAPP so converting this to web app will make it reachable to more clients, using browser friendly language like javascript a web app can be developed which will have nice graphics, multiuser capability and responsive with browsers used in any device. In this thesis I am going to do a comparative study of different approaches for porting a cross platform cocos2d C++ application into the web using different technologies like Emscription, Google Native Client and Flash etc. The purpose of this evaluation is to find a holistic approach in order to find the best solution for porting this application to the browser.

2 Application overview

TrunCAD GmbH was founded by Frank Rushmeier in 2004 specialized in planning, designing and pricing of the furniture using native application (desktop). Company have lot of products like 3Dgenerator, Truncad CNC, TrunAPP etc. I am working on prototype to check the feasibility of porting TrunAPP to web browser, currently it is only available in ios, android and windows platform, it is a cross platform application developed using cocos2d C++ with TrunApp one can enter dimensions and choose material for construction of the furniture. The App has drag and drop option with which one can drop different furniture structure on the canvas and can change their size, structure and material used. Once done then can send all the dimension, material details with the pricing via email. These dimension are latter on used by the clients for making the material cutting for the furniture and speed up the production process.[1]

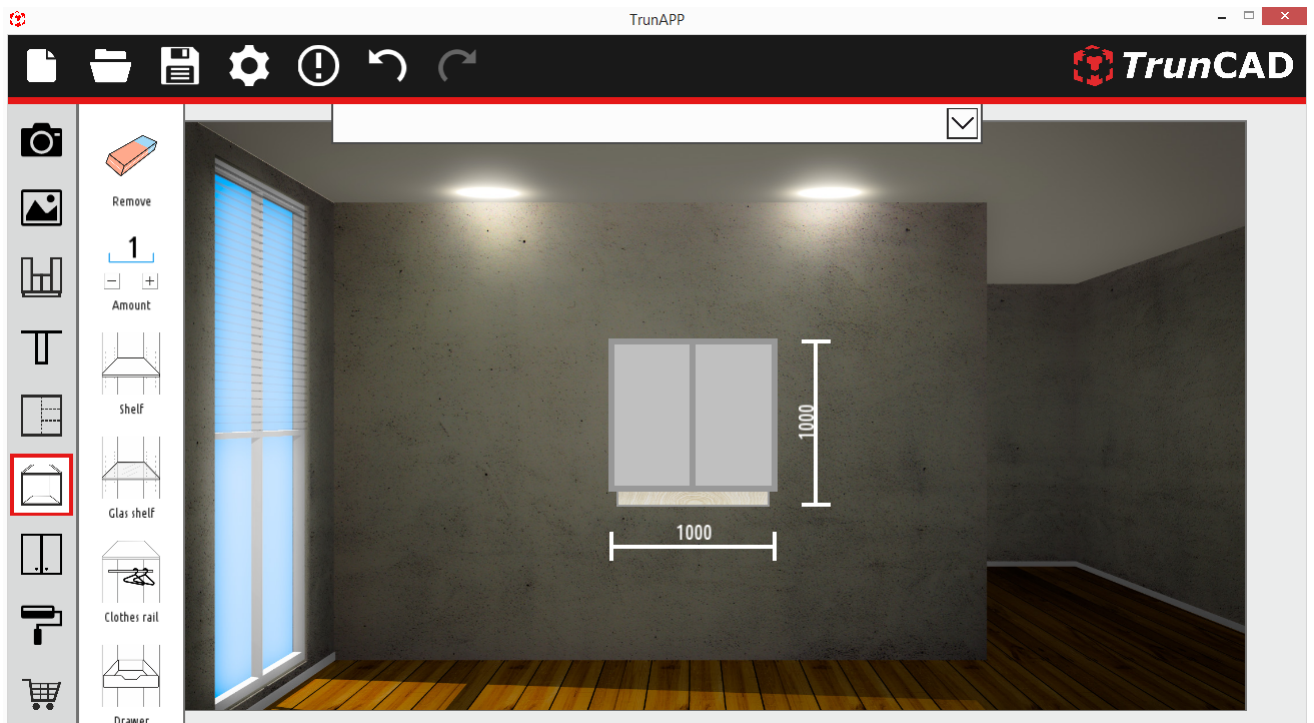
Description of the component of TrunAPP. Third: It mainly deals with encryption of data send.

Platform: It contains the pre-processor definition for camera, background configuration etc.

JSON: Contains files for reading, writing and utilization of json files.

App: It mainly contains the files like AppDelegate, firstLoadingScene and MainScene which form the base of the application.

Figure 1: TrunApp



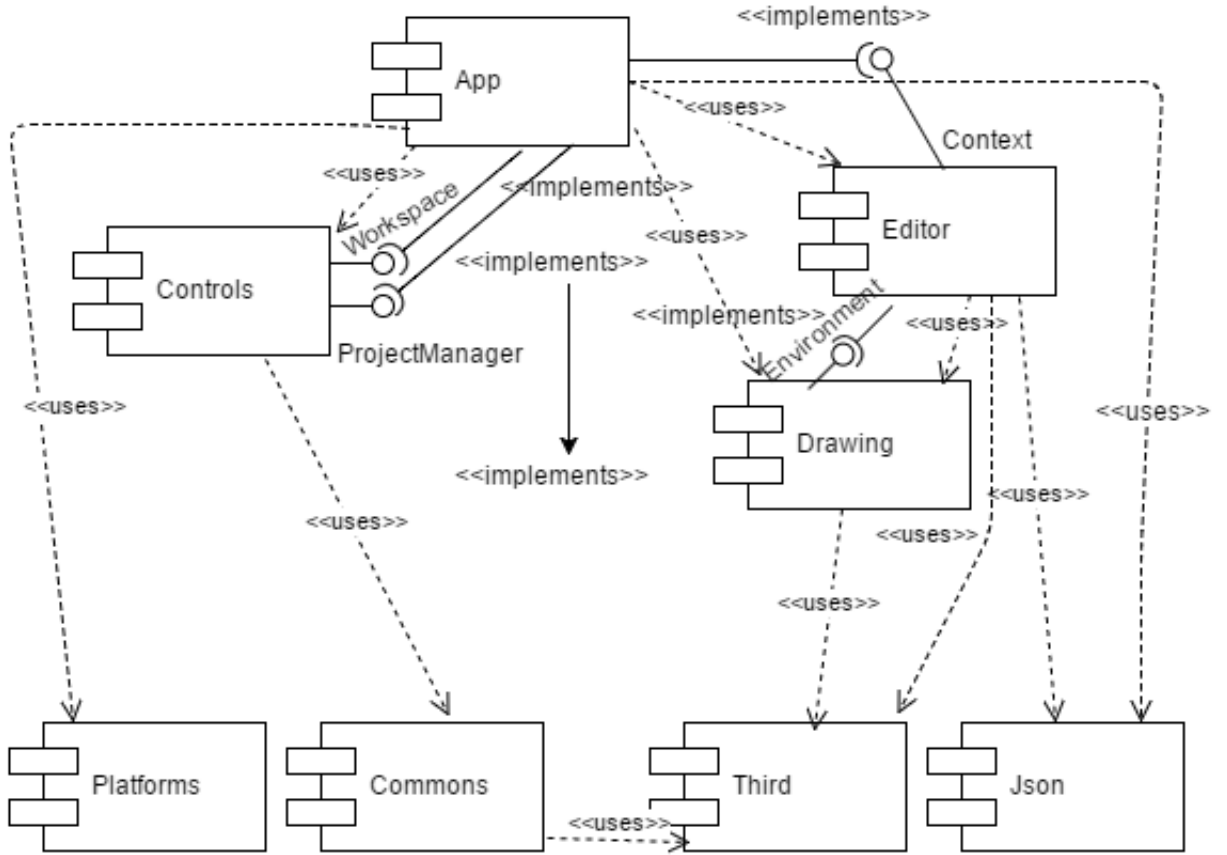
Common: It contains the files which mainly help in localization,utils,network, settings etc.

Controls: It contains the reusable buttons patterns, dialog box, labels etc.

Drawing: Contain files which deals with drawing parammeters like dimesion , field scaling , field pattern etc.

Editor: Files which deals with the editor setting scale segment , eit mode etc.

Figure 2: TrunApp



3 Background and related work

There are already some transpiler which converts C++ code to JavaScript. Below is the description of them.

3.1 Emscription

Emscription is a compiler which converts LLVM(low level virtual assembly) ¹ to JavaScript. Emscription itself is written in JavaScript

¹LLVM=Low level virtual assembly

and available under MIT license. Clang (compiler front end)² is used to convert the c++ code to the intermediate code (LLVM) code and then this LLVM is converted into javascript using emscripten. Emscripten is written in javascript the primary reason for that 1) the compiler can create javascript objects that represent constant structures from the original assembly code and convert them to a string using `json.stringify` in a trivial manner 2) the compiler can simplify numerical operations.

Emscripten has three main phases for its compilation:

1) **Intertyper**: Convert LLVM assembly into emscripten internal representation.

2) **Analyzer**: Inspects the representation and then generate the final phase including type and variable information, stage analysis and optimized data.

3) **Jsifier**: The data analysis and the final conversion to JavaScript is done here.

4)) **The runtime environment**: Some aspect of runtime environment of how it is implemented

a) Files sytem here are stored in memory.

b) Using HEAP array which emulate `malloc()` in C.

Relopper algorithm is the heart of emscription , it explains how the

²Clang = compiler front end

LLVM is converted to emscription blocks, for simplicity LLVM blocks are called labels and the emscription blocks as blocks. There are three type of blocks :

1)**Simple block**: One internal label and a next block which the internal label branches to.

2)**Loop**: It has two internal blocks one inner which represent the block inside the loop and next block which resides outside the loop and gets executed when it exits the loop.

3)**Multiple**: This block is generally for if, else, switch etc.

4)**Relooper algorithm follows this steps :**

It receives the set of labels and then does the evaluation of each label whether it is reachable or not if it cannot return back then it simply creates a block else if it reaches all entry then it creates a loop , if there are more than one entry then create a multi block. when the relooper process labels it replaces instruction accordingly for example when a block is created the instruction outside the loop are converted into break command and above are converted into continue statement and then it follows the semantics of JavaScript to get it go where it want to go. After that emscription does a pass in addition to optimize the code for example removing the continue commands that occur at the beginning of each loop. This way it helps to generate a JavaScript code that runs on browser. [2]

3.2 Google Native Client

Native clients help to provide computational performance of an application in web browser in a secure and safe way. Native Client uses software fault isolation and a secure runtime to direct system interaction and side effects through interfaces managed by Native Client. Native Client provides operating system portability for binary code while supporting performance-oriented features generally absent from web application programming environments, such as thread support, instruction set extensions such as SSE, and use of compiler intrinsic and hand-coded assembler. Combination of these properties in an open architecture encourages community review and integrating 3rd-party tools. When running NaCL application JavaScript components are first loaded these then helps to invoke NACL browser plugin to load the particular library into NACL container, when this module runs their behaviour is taken care by the NACL module for security purpose. For inter module communication native client use two option simple RPC facility(SRPC) and Netscape plugin application programming interface(NPAPI) , inter module communication also provide shared memory to avoid messaging overhead for high volume and high frequency communication. Native client use sandbox to detect the security flaws in untrusted x86 code such as self-modifying code and overlapping instructions this is generally achieve by prac-

tice of using aligned and structured rules and insures that native code module can be disassembled safely. NPAPI plugin also have support for interaction between service runtime and browsers, the service runtime maintains integrity of the sandbox and provides resource abstraction to isolate NaCL application from host resources and operating system interface, it also prevent untrusted code from inappropriate data access , sometimes additional measures like CPU blacklisting and NaCL whitelist can also be deployed. The service runtime also provides set of system services commonly associated with application programming which have system calls for both allocation and freeing of memory. [3]

NaCL SDK

Nacl software development kit includes a modified gcc toolchain that is capable of compiling c++ code to executable (pexe file) that can run in the web. The NaCL executable(naclsdk.exe) inside the kit helps to download the current or specified version Pepper API.

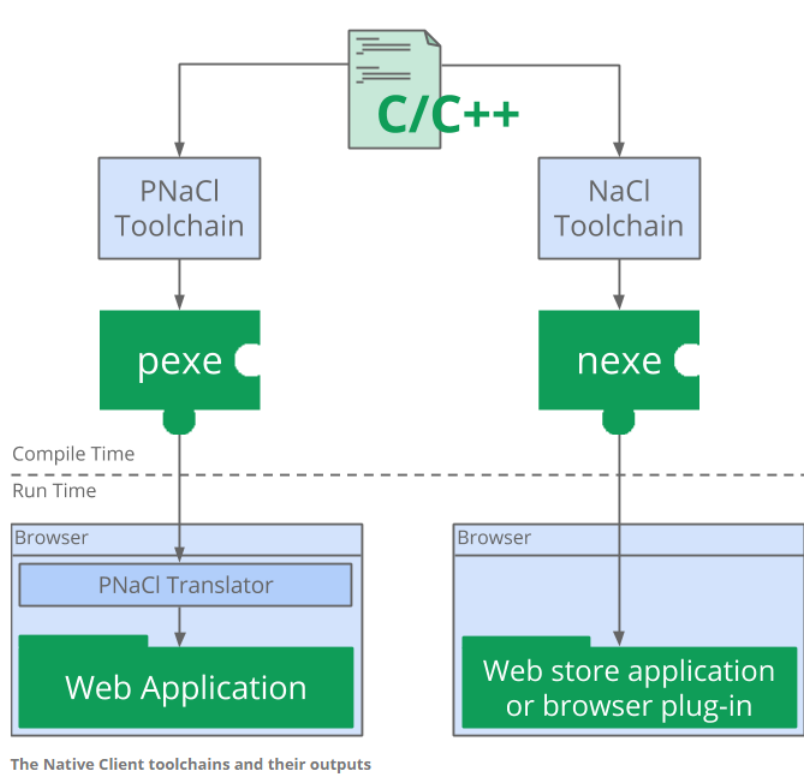
Portable native client

The portability problem is solved by PNaCL by splitting the process of compilation, first compiling the code into bitcode executable called pexe and then translating the bitcode to executable according to browser. These pexe is portable and can be used with HTML, CSS, JavaScript to run on user machine.

Pepper API

It acts as bridge and allow NaCL applications to communicate and pass data to the JavaScript side of the system, the data is generally transferred in the form of serialized text string format and then deserialized on the other end , Pepper API also handles event from mouse , keyboard and also create OpenGL context for hardware accelerated graphic. The data communication between web and the application is asynchronous which also helps to give good user experience.

Figure 3: NaCL toolchain and component interaction overview [22]



3.3 Cocos

Cocos2d-x is an open source game engine under MIT license .It is used to build apps,games and other cross platform application. cocos2d-x has a lot of powerful featur s and it can be coded in languages like c++,Lua and javascript. It is also cross platform and can be deployed into ios,android and windows.It uses the toolchain for developing multi-platform games with high performance. [4]

3.3.1 Cocos2d C++

Cocos2d-x C++ is used to build apps,games and other cross platform application.It structure consist of sprite,nodes,scence,layer, director,actions etc.Sprites are generally used to display an image on the screen.Nodes also known as containers also includes sprites inside them , this makes it flexible as by changing the aspects of the container its childrens aspects also gets changed.Containers have CC-Sprite ,CCLayer , CCScene which are inherited from the base class CCNode.CCScene contains one or more layers , it helps to break the applications into multiple scene for example in a game one for start screen , one for setting scene etc. CCLayers helps in creating CCMenu and other colored background like CCLayer-Color.CCLayer is often thought as CCNode that listen to user input such as touch,accelorometer etc and bundle all the protocol needed for it. CCSprite is the most important class that is used in cocos as this is repeated several time

in the application unlike CCLaeyrs and CCScene, it mainly consist of image that is inserted into the layers containers.CCDirector manages the scenes and knows all about the application. These what we discussed now is tip of the iceberg , other 90 % of the framework mainly consist of structures, macros and helpers.

3.3.2 Cocos2d-JS

Cocos2s-js is a game engine which helps us to create web application which Supported platforms include web, Android, iOS, Windows Phone 8, Mac, Windows, etc. It has high performance ,supports multiplatform development and user friendly. It helps to make 2D application very easily. It also includes cocos2d-html and cocos2d-x javascript binding in other word it can be said as cocos2d-js verison of HTML5. With cocos2d-js we can code and run games in browsers that support HTML5. Some of the features which makes cocos2d-js standout from other others is robust js API which runs anywhere dont require any plugin , easy to debug, high performance ,proven and with many new functionality including sprites, particles ,animations, timers ,transitions, action, , events (touch, mouse , accelerometer), persistence , file IO, sound, skeletal animations. In the meantime, Cocos2d-JS v3.0 comes with lot of new features like Editors Support, Assets Manager, Object Pool, JS to Objective-C/JAVA reflection, etc. cross platform

developers can also use cocos console for new project creation and can also boost their development using web version of engine and even deploy using the console , to get the hands on or casual game development one can use the cocos js lite version.Cocos2d-js with little modification can also run native application using cocos2d-x JSB , cocos2d-JS JSB is the wrapper code that bridges the native and JavaScript code, it also enables interaction between JavaScript and native code vice versa. [5]

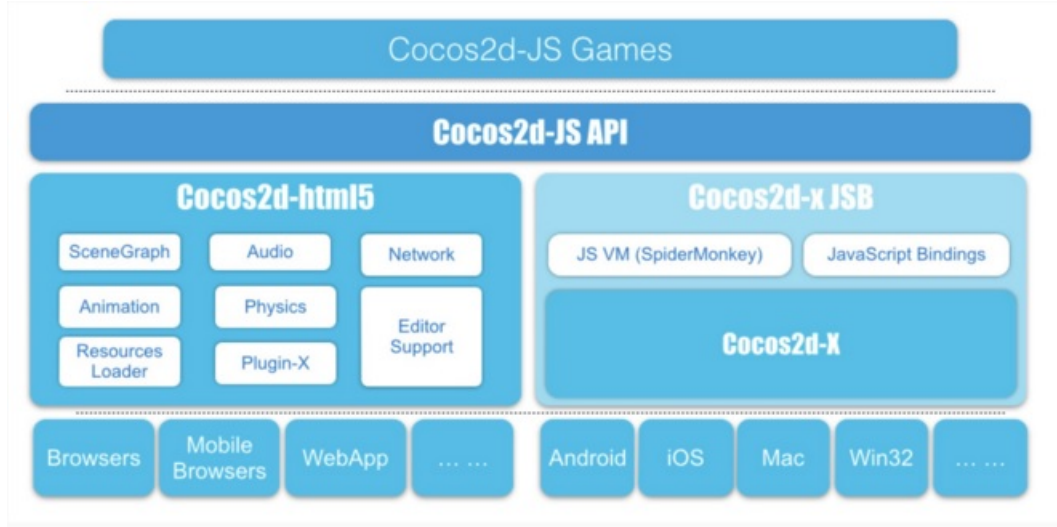
3.3.3 Cocos2d-HTML5

Cocos2d- HTML5 is the HTML5 version of Cocos2d-x project. Like other Cocos2d branches this is also an open-source web 2D game framework, released under MIT License. Cocos2d-HTML5 runs on browser which have HTML5 making it versatile as it can run on any platform. The backend of cocos2d-HTML5 is JavaScript this also makes it browser friendly and using API it also becomes completely compatible with that of Cocos2d-iPhone, Cocos2d-x JavaScript binding. [6]

3.3.4 Cocos2d JavaScript binding

Cocos2d JS is available with the same api for cocos2d-x, cocos2d-iphone and cocos2d-html5. While cocos2d-html5 will run on mobile

Figure 4: cocos hierarcial evolution for version 2 [14]



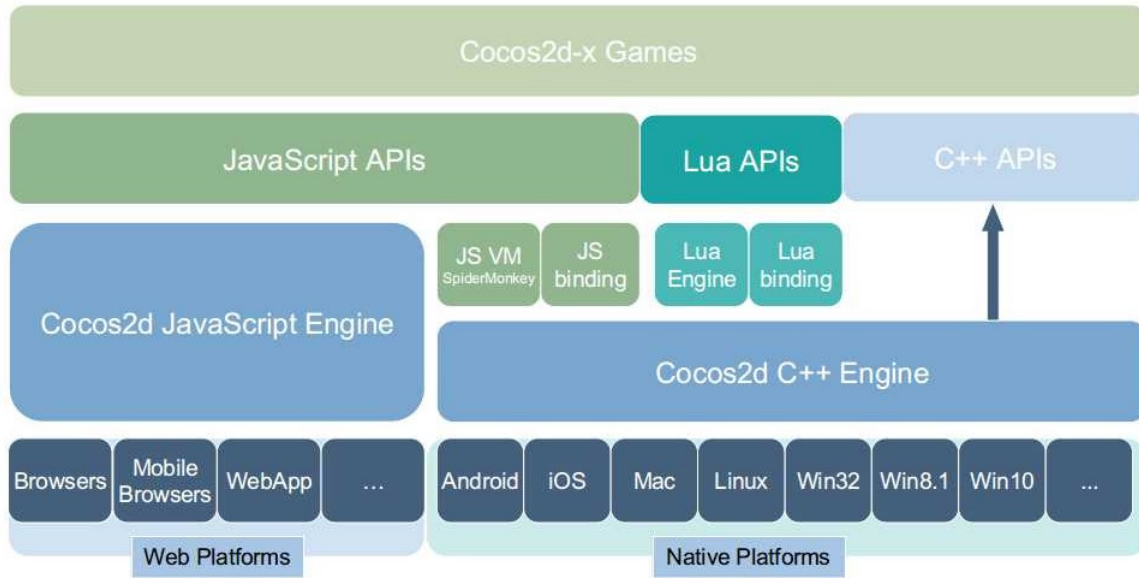
devices through the browser, JavaScript bindings have been created for cocos2d-iphone and cocos2d-x to drastically improve the performance. All graphics, rendering and physics code will run natively and only the game logic will run in JavaScript. Having the game coded in a scripting language provides more advantages than just the cross-platform aspect. As the code is not compiled, it can be replaced in run-time which allows much faster testing cycles.[7]

3.4 FlasCC

The FlasCC helps to convert C++ application to swf(small web format file) these swf ³ files are then embedded into the HTML which helps in rendering them on web browser which already have flash player(browser plugin) installed. CrossBridge is the open source ver-

³swf = small web format file

Figure 5: cocos hierarcial evolution for version 3 [15]



sion of FlasCC , it helps to bring native C++ app to web. Some popular game like neverball written in C++/OpenGL has been converted to web using FlasCC. [8]

3.5 CocosSharp

CocosSharp is .NET port of Cocos2D engine. It has most of its libraries in C # and F #. The core of CocosSharp is provided by the MonoGame framework, which is itself a cross-platform, hardware accelerated API providing graphics, audio, game state management, input, and a content pipeline for importing assets. CocosSharp is well suited for 2D games because of its abstraction layer. Furthermore, optimization outside the core libraries can also be done in larger games when it grows in complexity. In other words, CocosSharp provides a mix of ease of use and performance, enabling developers to get started quickly without limiting game size or complexity. [9]

3.6 Cheerp

Cheerp is open source compiler based on LLVM which helps to port C++ application to web. It uses the libraries of cheerp like cheerp/clientlib.h to do that, there is also a little difference from the normal c++ code for example the main () is replaced by webmain(). Cheerp creates JavaScript objects which maps with the c++ object, which makes the program more flexible and developer friendly. [11]

3.7 Compiler and cross platform compiler

A compiler is a program that converts one type of source code to other without changing the real meaning of the source code, the target code

is generally binary but can also be object code or intermediate code.

Following are the steps involved in compilation of this pattern

1. Lexical analyser.
2. Syntax analyser.
3. Semantic analyser.
4. Intermediate code generator.
5. Analysis and optimization.
6. Machine dependent code generation.

3.7.1 Lexical analysis

It takes the source code and transfer it into tokens. Tokens are generally the set of similar type of object for example set of keywords , operators and separators. These are generally done with the help of regular expression for example. Variable name:- [a-zA-Z-0-9]

Integer:- [+-]?[0-9]+

When choosing the lexer for variable and keywords priority is given to longer matches. First the NFA (nondeterministic finite automaton) is use to determine the lexer using regular expression , but this gets narrowed further using some more rules and restrictions and are know as DFA . Every DFA⁴ is NFA⁵ but the vice versa is not true. The states in NFA are reduced using the rule to small number of states

⁴DFA = deterministic finite state machine

⁵NFA = Nondeterministic finite automaton

of DFA. Say n is the number of states for NFA then $DFA = 2^n - 1$.
Lexer generator and sometimes inline code for lexer generation. Some popular lexer generators are Lex⁶ and Flex⁷.

3.7.2 Syntactic Analysis

It arranges all the tokens back not into text but into some structure, this typical data structure is called syntax tree. The node of these trees is the tokens found by the lexical analysis, if the leaves are read from left to right the sequence is same as the input text. Sometimes it is needed to arrange the format according to precedence at that point of time one need syntax tree as it forms the tree according to precedence.

Context free grammar

It is defined by the several set of strings, each set is denoted by a name which is called non-terminal , the tokens or alphanumeric character set are called terminals . $T=s | k$, where T is the terminal name and s, k denotes the non-terminal , as the operator is or so T can either be s or k .

symbol table

Symbol table is a table that binds names to information and for scope operations. The operations on the symbol table are accomplished by

⁶Lex = lexical analyzer

⁷flex = The Fast Lexical Analyzer

operation.

1. we need an empty table in which no name is defined.
2. Bind a name to the piece of the information , if any name already exist then the new one gets more precedence over the old one.
3. the list is searched in the lookup table till the match.
4. on entering the scope the old list is remembered and a reform is made to it.
5. on exit the old list is recalled.

As new binding are added to the front of the list , the list is searched from the front to the last and the binding in the inner scope will take precedence over the outer scope. so the worst case time for the lookup is proportional to the size of the symbol table. This is mainly the problem with libraries, it is quite common for a program to use the libraries that defines literally hundreds of names, to solved the problem names are hashed as into integers which are used to index an array. Each array element is then linear list of the binding of the names that is in the hash code. Using hashtable sometimes complicate entry and exit of scopes.

interpretation

Num = getvalue(num)

Id = lookup(vtable, getname(id))

Evaluation takes an expression and the symbol tables vtable and ftable and num and value which may either be an integer or Boolean. The

above expression shows the action needed to evaluate the expression. The pattern are on the left, it evaluate the result of the expression given as argument to end the expression.

It generally deals with semantics information from source code which include type checking.

Type checking

There are two type one is strongly type and the other is weakly type, Strongly type checking is to determine whether the argument of type are same as defined in the program. Example not concatenation floats with string. Weakly type language the type checking is not done, archetypical weakly types are generally machine code, and registers may be divided into integer, floating point address. Weakly type languages are mainly used for system programming when you need to manipulate move, copy encrypt or compress data without being concerned what the data represent. Some languages combine both strong , weak typing or both statistic and dynamic typing .Some types are checked before execution some after execution and some are not checked at all. For example C is a statically type language where no checks are performed during the execution and not all the items are checked For example one can store an integer type in a union type variable and read it as pointer or floating point number. Another example is JavaScript if you try to multiply two strings it will check if

the strings contain sequence of digit if so then read the string a integer and then multiply them.

3.7.3 Intermediate code generation

The final goal of a compiler is to run the high level code in compiler. Eventually the program will have to be expressed as machine code to run on the computer. From high level it is converted to Intermediate code then to machine code. Conversion of high level code to intermediate code is called front end compiler. Back-end compiler converts this intermediate code to machine code. The disadvantage of interpreting the intermediate code it is bit slower than executing the translated code directly. Some speed penalty can be eliminated by translating the intermediate code immediately before or during the execution of the program. Conversion of intermediate code is changing the expression into three or four address system which makes it easier for allocation in register , new variable are introduced using newvar and the translation of the variable and functions is done by vtable and ftable, that binds the variable and function names , transop translate name of an operation in the expression intermediate language .For some advance controls like goto , break/ext statement by setting a mark in the symbol table entry where the next statement jumps to .For array we need a variable to hold the base address of each entry.

The address is calculated when the array is allocated and then stored in the corresponding variable. Arrays are generally allocated on a stack and popped from the stack when procedure is called.

3.7.4 Machine code generation

The intermediate code that was discussed above is quite low level and similar to machine code, but machine doesn't have unlimited registers, therefore for mapping a large number of variables to small number of register is handled by registers allocation the simplest way of conversion of each intermediate into one or more machine code instruction.

3.7.5 Analysis and optimize

An optimization generally is about recognizing instruction that form a specific pattern that can be replaced by a smaller faster pattern of new instructions. For example if there is a code

$$X = 5 * a$$
$$X = 7a$$
$$Z = 5 * a$$

So, It can be optimized and reduced $z = x$ because both assign $5 * a$. Memory prefetching is also done to make the things more faster and optimized as it puts everything in the cache by declaring the amount of space in the pre-processing cache.

3.7.6 Memory management

Static allocation:- It means that all data is allocated at a place in memory , that has both the size and the address at compile time. Furthermore the allocated memory stays allocated throughout the execution of the program. So one can allocate space for say an array in the data space by placing a label at the current address pointer in the data space and then moving the current address by the size of array.

Stack allocation:- Call stack can also be used to allocate arrays and other data structures. This is done by making room in the current frame on the call stack and moving the frame pointer or the stack pointer allocation. The size of the array need not be known at the compile time, the frame at runtime creation makes enough space to hold the array. An unbounded number of arrays can be allocated as it is taken care by recursive function for allocation.

Heap allocation:- The limitation of the static allocation and stack allocation are often too restricting, sometimes one might want arrays that can be resized or which the survive the function invocation .Data that is heap allocated stays allocated until the program execution ends, until the data is deallocated by the system or declared dead by memory management.

Manual memory management:- Most operating system allows a program to allocate and free chunks of memory while the program

runs. In language C there is `malloc()` which allocates a block of at least `n` byte and return a pointer pointing to that block. If there is not enough space a NULL pointer is return.

Reference counting

Reference counting make it sure if no pointer is pointing to a present block in the memory and it is not accesed by a program then it memory can be freed. When a block is allotted, its counter field is set to 1 (it represent the pointer thats return by `malloc()`). When the pointer is added or removed it gets incremented or decremented accordingly when the counter becomes 0 then the block is freed by calling `free()`. It gets into some issues with doubly linked list, even if the last pointer of the doubly linked list disappears the element points to each other so the reference count will not become zero and the list is not freed. This is handled by not counting the back pointer which is sometimes referred as weak pointer it is not always easy for compiler to decide the weak pointers so reference counting is rarely used in languages with circular data structures.

Tracing garbage collector

A tracing garbage collector tracks which blocks are reachable from variable in the program and frees all the other blocks, by using garbage collector tracking circular data structure is not a problem. It marks the blocks as white, grey and black. White is not reachable from the set , grey itself is reachable but its childrens are not reachable, black

it means that both the root and its children is reachable, initially the root set is classified as grey and all heap-allocated blocks as white , when reachable analysis has been done all are declared as white or black. The white nodes are then freed. [16]

3.8 Cross compiler

When we develop an application, most of the time the development platform and the target platform (the machine where the program runs) are same, platform here means the combination of operating system and the CPU architecture. The process of building an application which is built in one platform and then runs on another platform is called cross compilation and the particular application as cross compiler. In case of interpreted languages like Perl , python as they can run in any target machine , cross compilation is generally needed for binary executable from source code written in compiled languages like C++. [13]

3.9 Basic difference between the structure of C++ and JavaScript code

JavaScript is a scripting language as by the name it looks its related to java but it differs a lot as java is a programming language while JavaScript is interpreted scripting language. Though JavaScript borrows many names and naming convention from java but its semantics are different. It is a multi-paradigm language that supports object-

oriented, imperative, and functional programming styles. JavaScript is generally used for client side interaction user control on browser application but advent of new framework like node.js popularity of JavaScript for server side scripting is also increasing. The only similarity between C++ and JavaScript is the look alike syntax. C++ is a statically typed, free-form, multi-paradigm and a compiled programming language. As compared to C, C++ incorporates object oriented features, such as classes, and other enhancements. C++ is now currently implemented on a wide variety of hardware and operating system platforms. It is object oriented languages and can be used to implement classes, inheritance and overloading etc. it is free for use same as JavaScript. There are some differences between JavaScript and C++ like JavaScript is a scripting language whereas C++ is a programming language, in C++ there is a code generation whereas in JavaScript there is none, in C++ safe casting, value types creation and value type variables passing as reference are possible where in JavaScript it is not possible, C++ supports constructor, destructor and finalizer but javascript only supports constructor, C++ supports multithreading and static variable whereas javascript doesnt support them which can be mitigated using multithreading.js (but it also have limitations All variables passed to functions must be JSON-serializable, threaded functions do not have access to the DOM, arrays and object are passed as value not reference) . [17] [18]

4 Evaluation of approaches

4.1 First Approach using Cocos sharp

4.1.1 setup

CocosSharp can be implemented via visual studio or xamarin studio, the recent version of CocosSharp v1.7 helps to make android,ios,windows application but no web support. For installing CocosSharp first one need to go to Tools — Options and configure the page Environment > Extension Manager and enter url <http://gallery.mobileessentials.org> and save. Once the Mobile Essential gallery is set up, one can go to Visual Studios Tools — Extensions and Updates... menu and a new Mobile Essentials node will appear under the Online category, where one can explore and install the tools, then click download and then install. Once done then Cocos template is installed after that C # project of ios,android and windows can be opened. [19]

4.1.2 Approach details

The reason for using CocosSharp initially was to find a method by which cocos2d-x c++ can be converted to CocosSharp using a wrapper and then to make a C # web application which can run on browser. But unfortunately the CocosSharp doesn't have the web application

building project in their framework.

4.1.3 Results

CocosSharp windows application project ran successfully.

4.1.4 challenges

Didnt faced any issues while compiling the project for CocosSharp it worked smoothly and without any problem, it compiled successfully an windows application.

4.1.5 pros/cons

Though there is a lot of similarity between C++ and C # code so the syntax of cocos2d C++ and CocosSharp were also somewhat similar but as it only support windows ,ios, android not web so this cannot be used as potential solution for porting the application into web.

4.2 Second Approach using Cheerp

4.2.1 setup

Download the cheerp installer and then run the wizard when run it will install cheerp in the following path c://cheerp//bin//clang++

4.2.2 Approach details

For converting the C++ application using cheerp library to javascript one need to include cheerp/clientlib.h header add client keyword in

front of the C++ function for making it compatible with cheerp for running in web browser. So when writing the code it is needed to do some changes in the code which makes it compatible for conversion by CLANG⁸. First trial was simple cheerp program which was successfully converted to web using clang but when tried with cocos libraries which were included in the code it gave error reason for the error is that it cannot including the header file because of the relative path. To solve the issue a makefile was made which include the source header file as the dependencies it still cannot detect the relative header file path. Another testing was done by building LLVM for that cmake was used, First llvm project code was downloaded and then was converted into .sln file using CMake linking CMake INSTALL_PREFIX macro to the cheerp folder and then tried to run the programme by adding Cocos in the additional libraries folder in project configuration properties in visual studio and then run the file but it gave error while the building. The next trial was to make an empty C++ project and the add both cheerp and cocos libraries to it and compiler as clang++ but it again gave error like cannot find header files. [20]

4.2.3 Results

For simple cheerp program without any external libraries it worked fine but when added with external libraries like cocos it gives bugs.

⁸CLANG = a frontend compiler for C++

4.2.4 challenges

The challenges involved is arranging the Cocos library in toolchain which can be used by clang.

4.2.5 pros/cons

Even though if it would have succeeded for cocos then also it was of no use as it needed to be in the format of cheerp library function to port it into JavaScript. Tried with one command of STL library but gave error as `cout` in `c++` doesn't hold any meaning here one need to use `client::writeline()` for that so this hold for cocos also for each DOM element cheerp have different syntax which completely differs from Cocos syntax in other words it will lead to recoding of the whole application.

4.3 Third approach using Google native client

4.3.1 setup

First one need to download the google native client SDK then unzip `nacl_sdk.zip` then have to go inside `nacl_sdk` directory and run `./naclsdk update` for windows and `./naclsdk update` for linux , it will install the latest version of nacl but if one wants to install a specific version then its needed to use the name of that particular version in command for update like `./naclsdk update pepper_(version)`, this will lead to installation of all NaCL dependencies.

4.3.2 Approach details

When NaCl SDK is installed and updated and its NACL_SDK_ROOT path is configured .Then build-nacl.sh script is executed from the top level of cocos2dx directory. This will build all the cocos2dx components and examples for all three NaCl architectures: x86-64, i686, and arm. But when the make file is run for samples it gives error as it cannot find the libcocos2d.a in nacl/newlib_x86_64/ Release folder, after downloading it from some sample project from github in the folder release it gave a new error "cannot find ldension and lchipmunk file in x86_64- nacl/bin/ld folder", tried to check it with pepper_canary but it gives build fails as it cannot find the files to build and then tried with removing the dependences on libcocos2d.a it compile well also made the bin folder for .nexe file of nacl which is inserted into html file for rendering but as it build depends on libcocos2d.a and the makefile root cannot find it so these .nexe file is not formed inside the bin folder. [21]

4.3.3 Results

NaCL run fine for simple C++ STL library application but dont work well with cocos as it is supported in the version 2.0 which is an older version of cocos(current version is 4.0) and files are missing in it which it needed for build.

4.3.4 challenges

During running the makefile file it cannot find the file in the location specified in the makefile so the build of the .nexe file failed

4.3.5 pros/cons

First of all the build failed for the formation of .nexe file and even if it somehow would have succeeded then we have to face two major problem one .nexe file is only supported in Google chrome and the second is the code needs to be changed according to pepper libraries for making it compatible for building of .nexe file.

4.4 Fourth approach using cocos2d-js

4.4.1 setup

Cocos html 5 is subset of JS , after downloading the cocos2d-HTML5 by cloning git directory one can run the `python -m SimpleHTTPServer` , this open a browser and go to `http://localhost:8000/projectname` , then changes can be made in the js file which are reflected into the canvas when the page is refreshed.

4.4.2 Approach details

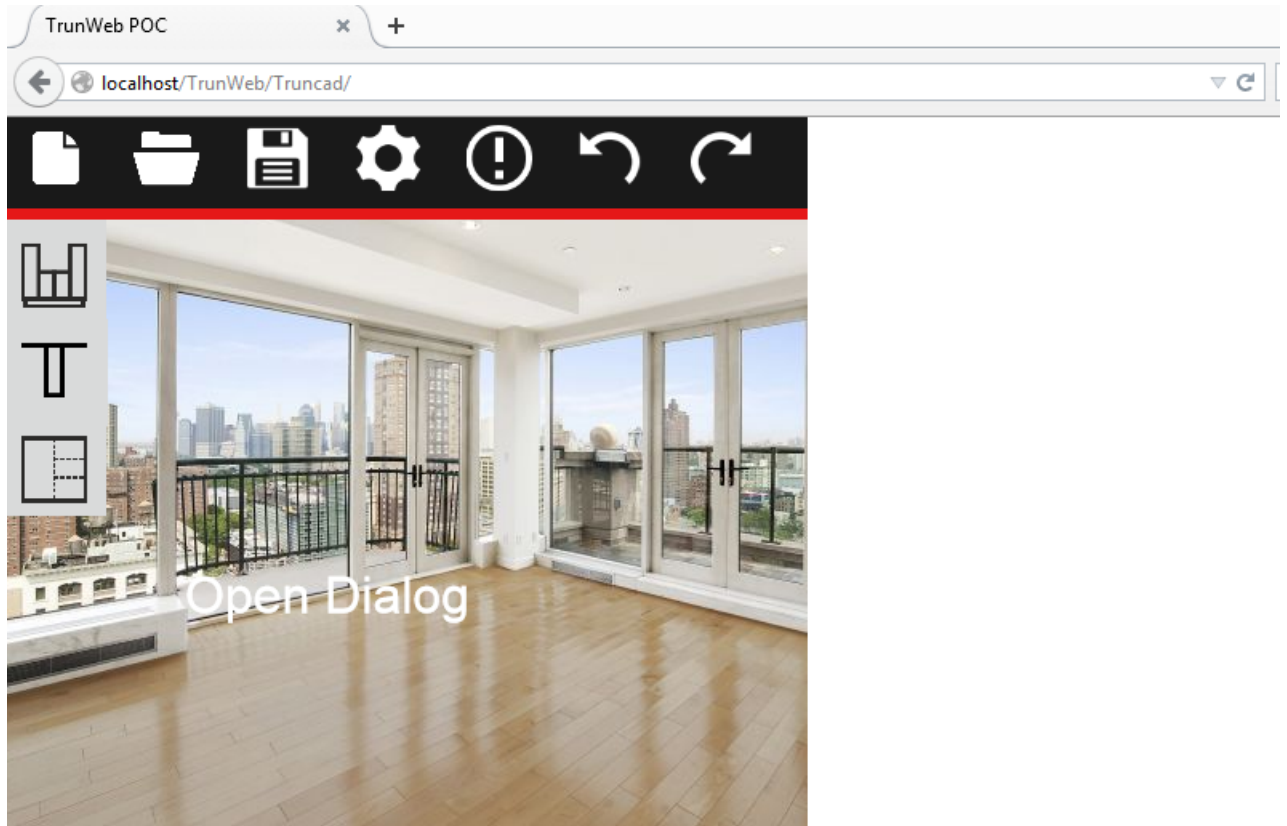
This approach form the base of the transpiler(Cocos2d C++ to Cocos2d JS), the cocos2d-js version was build first to see if it can be ported to web or not. Using cocos2d-js a small replica prototype of TrunAPP was made. Its compatibility was also checked with node.js,

socket.io it worked fine transferring the coordinates to the server. Cocos2d-js have some syntax similarity with cocos2d-C++ , which made me think of trans compiling the code. [23]

4.4.3 Results

Below is the result of web application made by cocos2d-js.

Figure 6: cocos2d-HTML5 running on web server



4.4.4 challenges

There were no challenges involved in recoding it with cocos2d-js

4.4.5 pros/cons

Cocos2d-js became the base of transpiler research and a good option for porting the existing code into web.

4.5 Fifth Approach using emscription

4.5.1 setup

First one need to download the Emscription SDK web installer , web installer is better than portable sdk zip because here it automatically installs and configure the whole Emscripten toolchain (Clang, Python, Node.js and Visual Studio integration) in a single easy-to-install package. For installing in Linux or mac platform its needed to download the zip folder from the emscription website and then unzip it and run the linux command to install all the dependencies like python,node.js,git,cmake,jre and then downloading fastcom(LLVM and CLANG) via git and then configuring the build using Make, once this is done then ./emscription file has to be run to configure the compiler setting. [25] [26]

4.5.2 Approach details

Following steps were followed to port cocos2d C++ to JavaScript first coco2d-x repository was cloned and then the sub module updated once thats done its needed to enter inside the cocos2d-x tree and then run the makefile called makefile.emscription which leads to compilation

of all the samples inside , Once this is done, one will find .js, .data and .html files in the proj.emscripten/bin/release directory. Datafiles contain packed versions of all the assets needed for the game but in my case because of the missing file it did not compile well. One thing to note is that this is only available in cocos2d git version 2 so one needs to checkout v2 to get the older version. For windows it is needed to directly run it from command prompt with path of the makefile. For using it with visual studio, emscripten is integrated into Visual Studio (2010) using the vs-tool plugin, which is automatically deployed by the Windows NSIS Installer Emscripten SDK if Visual Studio 2010 is present on your system at install time (it is also possible to manually integrate Emscripten into Visual Studio 2010). The vs-tool plugin adds an emscripten configuration to the list of available Solution Configurations in Visual Studio. Activation of this configuration for a solution/project to make Visual Studio run the project build through Emscripten, producing .html or .js output depending on the configured project properties. First selection of Emscripten configuration from the Configuration Manager dropdown as the active configuration. Then building the Solution to launch the project directly to a web browser from Visual Studio by creating a new environment variable "EMCCWEBBROWSEREXE" and setting it point to type of browser for opening web pages generated using Emscripten via Visual Studio for this one needs to right-click on the project to run, choose set

as startup project, and select Start without Debugging (Ctrl+F5). This should launch the generated .html file to the browser, but in my case it was giving error cannot find the gcc. The thing to note is that it only build on visual studio 2010 is possible not 2012/13 as they dont have FASTCOMP.

4.5.3 Results

when running OpenGL code it compiles well but with some error but it takes a lot of time to compiled a simple graphics programme and in case of emscription with cocos as dependencies it gives error for both windows and Linux as the files needed for it was missing.

4.5.4 challenges

when running emscription with cocos as dependencies it gives error like for linux it gives error cannot find the emcc file as it was missing in that version tried to download that file and then again build it gave the error and the build failed. For windows it gives macro error and for visual studio integration the build fails giving an error The "GCCCompile" task was not given a value for the required parameter "GCCToolPath".

4.5.5 pros/cons

Emscription was the only approach which doesn't require recoding but as the new version doesn't support it and the old version has some missing files so this approach was unsuccessful.

5 Validation of the approaches

5.1 Comparison of the approach

Automatic generation of code: - Of all the technologies mentioned above except emscripten and the transpiler all others need to rewrite the code using the library of that particular technology. Example cheerp we have to include `cherrps client.h` and `clientlib.h` for making it compatible for transpiler to change it into javascript code, `client::console.log(hii)` is used instead of `console.writeline(hi)`, in cheerp the main is replaced by `webmain()` it creates the binding and helps to create the JavaScript object when the following cheerp command is run `clang++ -target cheerp example.cpp -o example.js`. Another example is native client. Where it's necessary to have three components a factory function `CreateModule`, a module class `pp::Module`, an instance class `pp::instance` and also to use these functions we have to include `ppapi` library, [46] then only we can form the `.nmf` file using `make` and `pepper C Api`. Same thing also happens with `flascc` where one has to write binding code between `c++` and `actionsript` to run

it as plugin in the web browser for example we have to include `<AS3.h> inline_as3(the expression inside);` and then execute it using `gcc -emit-swf test.c -o test.swf` to get the swf file to run on the browser [31]. Emscripten run directly the c++ code to JavaScript using LLVM+Clang these JavaScript code generated is in the form of `asm.js` which is bulky and store lots of thing in heap and consumes lot of memory. When compiling a openGL code which have graphics it takes a lot of time as it optimize the thing it also leads to lot of error while compiling , it creates code which are not so easy to understand as it creates the element into the heap(`asm.js`) example `var utf32 = HEAP32[(((ptr)+(i*4))> >2)];` this also creates issues when something needs to be changed. Transpiler converts the `cocos2d-x` c++ code to the `cocos2d-js` these helps to form JavaScript objects ,so you can easily rewrite it when any small changes is made and its completely under the control of developer. As the transpiler just do the source to source conversion so the time taken is less. [50]

Web browser support: NACL is supported only in Google chrome it is sandboxing technology which helps to run the application on browser with near native code speed. But when run on other browser it doesnt run anything. It faces a lot of critics it is not liked by Mozilla (Firefox) because of the interoperability issue so they dont have any plan for implementing this. Opera also doesnt support the idea because of the security issues and complexity.so making web application

using NACL will have its browser limitation [46]. Same with flascc before running it any browser the flash plugin needs to be installed if it is not already there, then only the flash runs and also it takes some time for the initial loading of the file if the flash file is bigger. These are big turn off for that they have to install a new application flash to run it on web [49] [48]. Whereas the web application made with cocos2d-js and emscripten runs on all browsers because of JavaScript which doesn't require any plugin. The code made with cocos2d-js runs as native code animation loading is fast and it also consumes less memory whereas the code generated by emscripten having JavaScript in asm.js pattern memory requirement is high. [45]

Portability: NACL and PNaCL is portable as it is adjustable to the new hardware specially the processor and once .peexe file is produced it can be placed in html and can run easily on Google chrome in any operating system and any processor. [46] Flash it is completely compatible with cross browser as well as hardware as it only needs the plugin to be installed and any browser, it starts running. Emscripten creates portable code which can easily run on browser except few exceptions like code that have multi thread and uses shared state, it is all processor dependent the code which runs on x86 might give run time exception in ARM because of address read and write instruction. [29] Code which are specific to the certain architecture generally used inline the code like `asm()` for x86 need to be disabled when running on

other processor or can also be optimized to match a certain processor instruction set requirement. Where the code generated by transpiler cocos2d-js is form of JavaScript which have object so it easily runs on any operating system and any browser as it doesnt have any inline code which is processor or operating system specific.

Convenience: It is very convenience to implement NACL as you need to only embedded the .pexe file in HTML and when implemented it is also very easy for client to use as did not need any plugin, but the pain of recoding it according to ppapi libraries still remains. It is also very convenient to implement flash as once the .swf file is created it can be easily embedded in html and can run on any browser which have flash installed, but writing code in actionscript libraries is sometime cumbersome .regarding the code generated by cheerp the developer need to write the code according to library of cheerp which is little painful and time consuming , but the JavaScript code is generated it can be easily run in the browser without any issues.The code generate by emscription is very easy to implement as one need to run the emcc to convert the c++ code to javascript but it something gives a lot of error when converting and as well takes a lot of time to compile so it is time consuming if one have to rerun the code just for some small change. Where cocos2d-js which is tranpilied can run easily on browser and can also be updated easily , just some small correction needed to be done after the recoding if it doesnt work as no transpiler

can convert the code 100 % .

Security: NACL is secure to use as it has limited access to the system because of sandboxing so it provides the same level of security as any other web application on browser [28]. Flash which highly depends on swf has some vulnerabilities like malicious data injection for example flashvar in url that points to another swf the attacker can point to another malicious swf, swf is also vulnerable to spoofing attack. Insufficient authorization in swf can lead to cross domain attack and data theft when it is hosted in a shared domain [33]. Where in emscripten where c++ code is converted to subset of javascript(asm.js) are typed array format without any sandboxing to protect it from the outside threat. cocos2d-js faces the same threat as javascript faces like cross site scripting, cross site request forgery. The way the javascript interacts with HTML DOM items really makes it more vulnerable, as it can be easily used to insert malicious script. There are two ways of making it more secure first by sandboxing which gives permission to access limited resources and the other making some policies which restrict the access to other domain [51]. These issues can also be solved by vulnerability checking in certain time interval by using security analysers.

5.2 Evaluation of comparison result

As discussed in the above comparisons between the different approaches we see pro and cons of applying a particular technology, now we will analysis which technology to use to make the web application, as per the thesis the cocos2d-x c++ application needed to be transferred to the web browser without recoding so here all the technologies that we saw NaCL, cheerp,FlasCC all are needed to be recoded using their respective libraries and then can be compiled to the JavaScript code , here we have an existing application in our company called TrunApp which have already been done in cocos2d-x C++ so it would be cumbersome and time consuming to recode these using other third party library.Only emscripten and transpiler can help in porting , emscripten is a good choice but have some really serious issues as discussed above like creation of asm.js a format javascript code which is memory consuming and as well it doesnt compiled the cocos2d-x C++ code to JavaScript giving a lot of compile error which is hard to correct. So in that case the transpiler is the right choice as it help to compile the code to cocs2d-x js which has its own javascript object and can easily updated with small changes as well it would be in total control of the developer as coco2d-js is a proper user friendly, open source programming language with cocos community support. The transpiler that does this is made with C # and have the facilities like training the

data for new syntax so it is also flexible to the version change in the framework.

Figure 7: Summary of comparison of approaches

Approach Name	Advantage	Disadvantage	Result	Remarks
Native client(NACL)	it helps in porting c++ App to web	only works in Google chrome	Not successful	Gives a lot of bug when compiling for cocos
FlasCC	it helps in porting c++ App to web	makes flash plugin	Not successful	crossbridge(Open source) doesn't have the flasCC
Cocos Sharp	It helps to make application for android, windows, ios	It doesn't port c++ to web	Not successful	Not useful for porting
Emscription	Its helps to port the c++ to JavaScript, works in all browser	While trying to port cocos to web it give error	Partially successful	Potential solution
Cocos2d-HTML5(using cocosjs)	It helps to port c++ to JavaScript but one have to recode	recode	successful	Can be made easy by transpiler

5.3 Evaluating feasibility of code for transpiling

There are lot of similiarity between between Cocos2d-x (C++) and Cocos2d HTML5 (JavaScript). In fact most of the difference can be solved by using find and replace: just look for the CC prefix and replace it with cc. (that is cc and a dot). Looking for specific data types and replacing them with Js data types like var. Some replacement like finding the string :: and - and Replace with . (dot).

Example

Listing 1: Similarity between cocos2d-x c++ code and cocos2d-js code

```
//C++

_bgLight = CCSprite::create("bg_light.jpg");

_bgLight -> setPosition(ccp(_screenSize.width * 0.5f, _screenSize.height * 0.5f));

//JS
this._bgLight = cc.Sprite.create(bgLight);

this._bgLight.setPosition(cc.p(SCREEN_SIZE.width * 0.5, SCREEN_SIZE.height * 0.5));
```

so it can be easily translated using semantic and syntax tree and then the can easily be optimized. [32]

6 Process of Prototyping

6.1 Introduction of Transpiler

Transpiler also known as source to source compiler, it converts source code of one language to other , the only difference between this and compiler is that transpiler converts the code of one language to other following the same level of abstraction while compiler converts it from high level language to lower level language(machine code).Transpiler has lot of similarity with the language translator which divide the

words into verb, noun , adjective tokens it also divide the code into keywords, type and operator. Collection of similar type of entities makes it easy for search. Transpiler like coffescript transpiler, pseudo-master(A restricted Python to idiomatic JavaScript / Ruby / Go / C # translator) uses the same first few step of compiler for transpiling. It takes the source code and then using lexical analyser divide it into tokens these tokens are then converted into a syntax tree and the tree grammar is then checked with the particular language grammar, if its syntactical way correct then it is translated to other language using the lookup table [37] [55]. For language translation lot of algorithms are used like Hierarchical phrase based model where the next translation depends on the previous subset and the next subset ,synchronous CFG also known as syntax directed transduction grammar , denoted by the rule $X \rightarrow \gamma, \alpha, \infty$ like hindi to English translation $X \rightarrow \text{kyun, why}$ [56]. Some transpiler like pseudo-master ⁹ which translate the code into simple English use rule base approach which uses detailed information closely related to the structure of the source code and are able to generate code when their rules matches the given data. In data based approach the text of the code is retrieve from the training data. Here the accuracy can be increased by training the data more or expert advice on the data. Statistical machine translation it is based on both rules:

⁹pseudo-master = A restricted Python to idiomatic JavaScript / Ruby / Go / C # translator

1. Extract the relationship between first two languages input and output languages.
2. Using this relationship to synthesis new sentence using statistical method to decide which translation is the best.

The foundation of SMT mainly consist of phrase based machine translation (PBMT) and tree to string machine translation (T2SMT). Phrase based translation generally uses phrase to phrase relationship between source and target language for system.out.println of java to console.writeline in c #. For example $p = [\text{if } x, \text{if } y, x, 1]$. Head insertion also known as the root node followed by the leave node heads become the path way for easier translation as all the head of the source code are kept on the left and all the head node of the target languages are kept on the right.

2. Pruning and simplification: in this the stage pruning of the tree is done with some handwritten rules by removing nodes that doesnt seam related to the surface of the language. Then training of data take place where the parallel corpus is made where corpus contains pairs of the source statement and there corresponding target code. For automatic evaluation of one use BLEU(bilingual Evaluation under study) it uses the length n word generated by machine and then one feed by the user , this is known as n-gram precision , BLEU value ranges between $[0,1]$ if the translated result is equal to the references then the score is 1.

Other source to source compiler cetus, it is a source to source restructuring compiler infrastructure for C and followed on project to polaris translator. cetus follow class heiracial structure 1. Symbol table: cetus table provides the information about the data types and identifier. All the cetus IR classes are derived from traversable it traverse the list of object in generic way. Annotation: pragmas, comments, directives of the IR objects are stored in annotation objects. Cetus follow symbolic manipulation by simplifying and normalizing the expressions like $1+3*4-5+a=8+a$ this is known as folding. cetus array section mainly consist of use/def pattern, before the array analysis applied the arrays are simplified. Ex. `pragma cetus use (A[0:100]) def(B[1:99])`. it also uses data dependency framework to push values into the variable for example for loop : `for(i=lb; i< ub; i++)` where the value of lb and ub are given by analysing the data structure of the code. The transformation phase consists of privatization, reduction variable recognition and induction of variable substitute. In privatisation algorithm traverse the loop from inner to the outermost loop to decide the private variables. Reduction variable recognition is done by criteria to define what type of variable it is. Some of the features like automatic translation and compiler driven optimizations makes it widely applicable to source level optimization, adaptive runtime tuning and transformation of both multicore and large scale parallel program. Transpiler like coffescript which transcompiles to

javascript most of things in coffescript are in the form of expression which returns a value. Few basics which makes coffescript coffescript syntax much easier than JavaScript are no need of semicolon, instead of curly braces indentation are used , whitespaces are used to delimit the codes and parenthesis are not used for passing the arguments. The coffescript parser is generated by jison parser, jison sytle is similar to Bison and its is a bottom-up parser used for implementation of javascript.To create the jison parser we list to match the pattern on the left side and steps needed to create the syntax tree nodes on the right side. As the parser runs it shifts tokens from token stream from left to right to get the non-terminal nodes. The grammer of coffescript is generally defined by the name of non0terminal as key to the list of patterns with each match action dollar sign are provided as its value.grammer for precedence goes according to the BODMAS rule. Tokens are generated using regex from the source code, after the match found tokens are stored and then the parser move ahead to find another token. Tokenizing also follow some rules of grammar and also checks to ensure that keywords are not used as identifier.tokens that are used in coffescript ae keywords,identifier,number whitespace ,comment,multident,string,compare , math , conditional , relational,bool etc. Rewriter also can be called as look ahead and look behind algorithm , it rewrite the tokens by looking one token ahead and one token behind giving the idea how many tokens to move forward or

backward in the stream these helps the parser to not miss any tokens .rewrite helps to make the clean the things up which are result of complicate grammar and bloated parsing table . Syntax tree in coffescript is form considering the grammar nodes.coffe contains all the classes for the formation of nodes, then compile command is fired to convert the tree into JavaScript string. All the nodes are then converted to code fragment objects, it contains the details from where the code came these code fragment are then combined to form code snippets. Base is the abstract class for all nodes in syntax tree , the subclass of this base class implements code generation by executing CompileNode method. The scope class regulates lexical scoping within coffescript when declare it form a tree of scope each scope knows the variable declared within and as well the reference to the parent scope, like this we understand with variable are new and which are needed to be declared using var. All these lexing, rewriting and node formation is done using helper these classes contains the common utilities that are used by the by these classes for example counting characters ,sorting, merging ,trimming , count the no of occurrence of string etc.

6.2 Observation of the code structure of cocos2d C++ and cocos2d JS

There are lot of similarity as well differences between the two languages. To name a few JavaScript generally use var for the variable

of any data type which have some resemblances with auto which also can fit with any data type, in cocos2d C++ there is # include for the header files but here in cocos2d js the files are generally loaded in cocos2d.js using loadjs(filename.js) function. In cocos2d C++ MenuItem position should be set, but in cocos2d js that is not required it takes the position of the menu. As C++ is language which get compiled whereas JavaScript is interpreted, so it doesn't follow the same structure when getting build as function which defined below the function call in C++ get recognized whereas in case of JavaScript it gives error that function is unknown as it is interpreted.

6.3 Architecture of the transpiler

The transpiler that build by me is in language c # it follows the following steps for transfer of one source code cocos2d-x C++ to cocos2d-x JS source code, first it takes the lines from the source code and then convert it into tokens like keyword (the reserved word in a language), operators (=, +, - etc.), separators like ;, etc., identifier (regular expression for alphanumeric). After separator of code in into tokens it becomes easy to search and translate a particular token as they are in form of collection. Parsing of a particular line is done where when the token is detected then that word is excluded and then the rest of the string is parse with the help to check for tokens like keyword,

operator, separators and identifier.

Listing 2: Algorithm Lexical analysis

```
Input: line
I=0
For I< line length
    Convert line to char array ch[]
    If(checktokens(ch[I]))
        Then line length = character array length
    Else
        Continue the character array traverse
Increment I
Output: line in form of tokens
```

Second step is checking the grammar to make it sure that the code that we are transferring is target code is correct. Grammar is checked first in NFA¹⁰ pattern where it checks the pattern like after a keyword what should be the next token type this is done by look ahead algorithm which checks for such pattern in the code by doing one or two tokens ahead and detecting the pattern provided in the grammar to make it sure that the grammar is correct. Mostly regular expression and stored array are the method used for that detection.

Listing 3: Algorithm for grammar check

```
Input: line
i=0
For i< number of tokens in line
```

¹⁰Nondeterministic Finite Automata

```

Get the first word
If(check token(word))
    Check the grammar pattern for that particular word
    Look ahead algorithm
    If(line[i+1] equal type of token AND line[i+2] equal type of token)
        Correct grammar so continue
    Else
        Error in grammar
Increment i

```

The code is parse from left to write and in this parsing it forms the tokens checks the grammar and translate the code, the code translation is generally done using one to one mapping of the tokens that are well trained by the training algorithm which we used here, this algorithm check the probability of the of the particular word using the regular expression and pattern recognition to decide the confirmation of a particular word for that particular token. The target translation word comes from an xml which have token name, source word, target word and weightage. The weightage for a particular word decreases if a mismatch is found.

6.4 Method and algorithm used

6.4.1 Hidden markov model

Hidden markov model use the previous data of observation from history and then using the probability calculate the next step data and

these are sometimes derived by using matrix multiplication of some more related probability data. Below is the example to make it understand better. For example there are only two temperature in a year Hot and cold.

H C

H 0.7 0.3

C 0.4 0.6

Above matrix denotes hot year after hot year is 0.7 and cold year after a hot is 0.3. suppose current research show a correlation between the size of tree growth ring and temperature. where S , M , L denotes small, medium and Large respectively.

S M L

H 0.1 0.4 0.5

C 0.7 0.2 0.1

When the temperature is hot the probability of a tree with small tree size is 0.1. If the initial distribution is $A = [0.6 \ 0.4]$, by multiplying these probability we can get a approximate idea of whether in coming next year.

6.5 Implementation of the transpiler

For training the system probability was used the variable which decides this are kno means the number of character in a token and other is kreg which is closeness of a token to a particular pattern. Average of this probability helps to determine the word is a token of what type. Initial value is feed of kno and kreg are determined from the word feed into the system. After observing the patterns of the tokens one can easily make a certain guess what token type it represent like keyword are generally 4 letters , operators and separators are single letter and identifier more than 5 words. So to get the probability we divide the (number of letter token type/the word length) if sometimes the value becomes more than 1 which shouldnt happen in probability we replace it with 0.1 making it as outlier. Then the word is again checked with regular expression for a particular tokens type to determine its proximity to that type and assign the probability value to 1 if its matches perfectly else 0.the average of parameters help us to decide the type of token. Then the word is again checked with pattern of the token for a particular tokens type to determine its proximity to that type and assign the probability value to it 1 if its matches perfectly else 0.the average of parameters help us to decide the type of token. These values are then feed into training xml file with attributes like

Listing 4: xml structure for training

```

<Translation>
<type>keyword</type>
<name>auto</name>
<tranName>var</tranName>
<weight>1</weight>
</Translation>

```

Weight of the particular type and name is decrease if there is no match, these helps Use to determine the outlier and get the perfect matching translation.

When the code is passed though lexical analyser then it gets divided into tokens by using the values from trained xml. The code is then checked grammatical correctness using look ahead algorithm and hidden markov model. Then it goes through the translation phase where first it is check against the peg grammar where the phrases of the code are arranged in regular expression format and then they are replace with the translated word it helps in translation of functions , class and cocos2d-x C++ elements.

Listing 5: Example of peg grammar

```

Regex regex5 = new Regex(" auto (.*) (=) (.*)::create ();");
var v3 = regex5.Match(str);
string s3 = v3.Groups[1].ToString();
if (!string.IsNullOrEmpty(s3))

```

```
{  
    string st2 = "var " + v3.Groups[1].ToString() + " = " +  
    v3.Groups[3].ToString() + ".layer()";  
    return st2;  
}
```

the order of flow is first the system is trained then the code is broken into tokens and the grammar is checked and then it is translated, below is the sequence diagram which shows in details.

Figure 8: Sequence diagram for transpiler

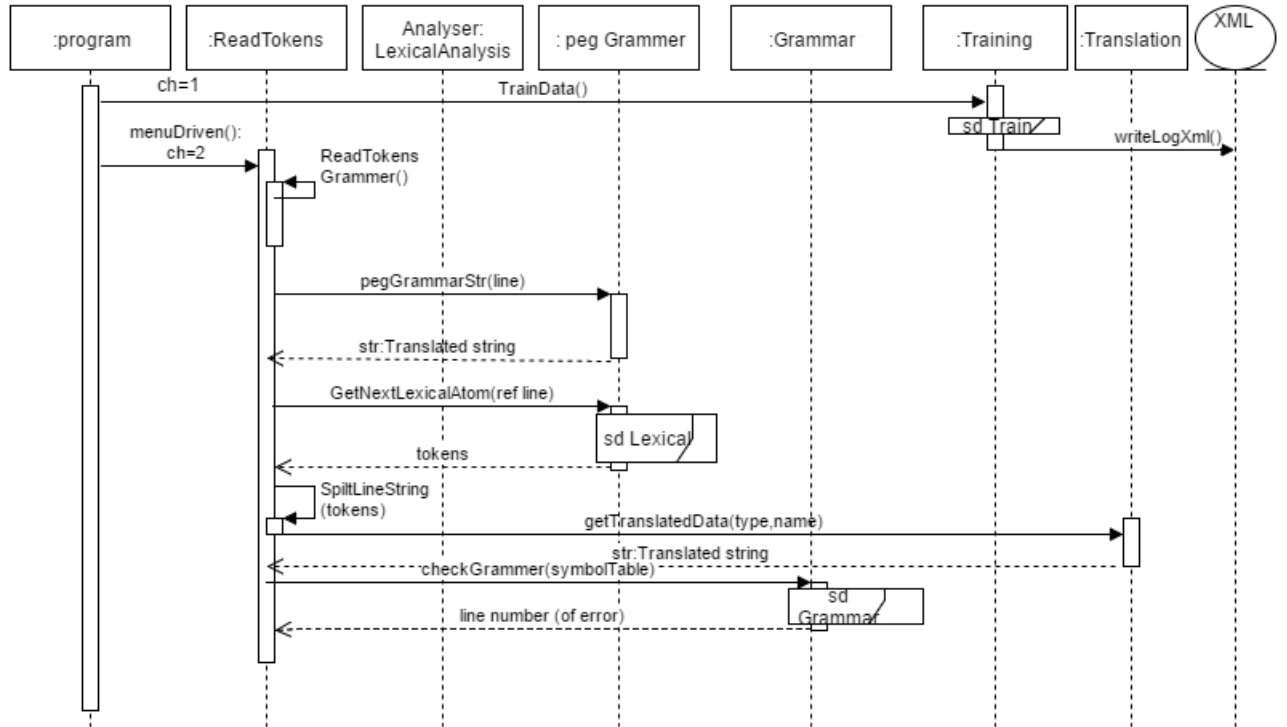
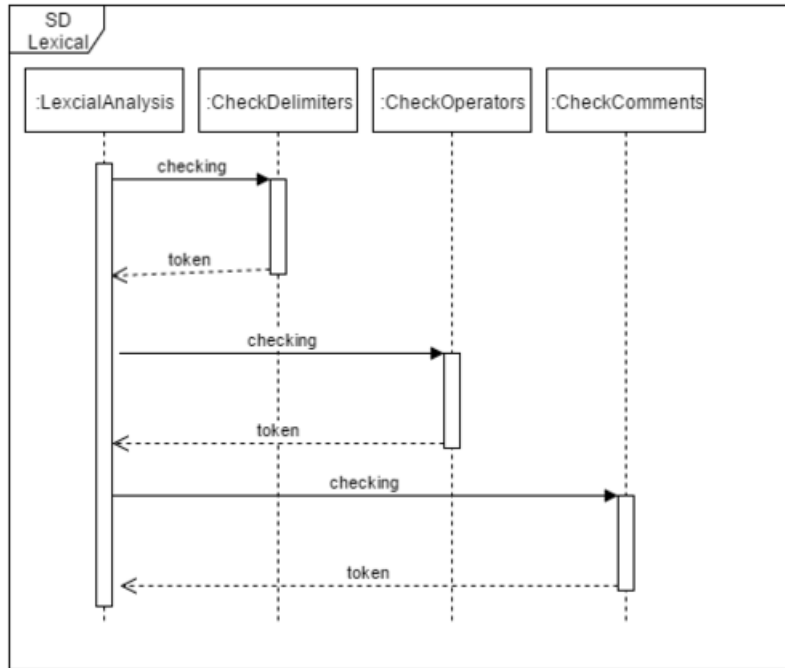


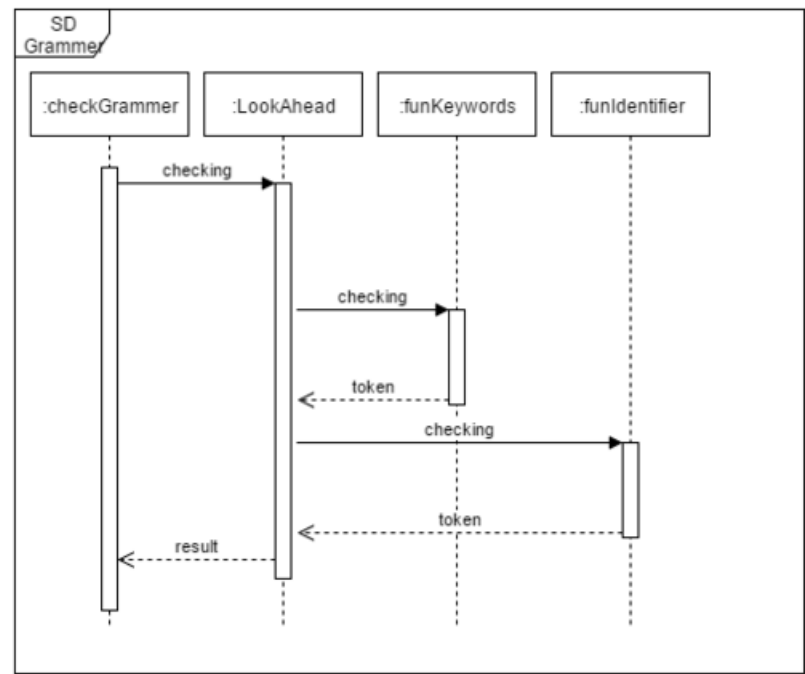
Figure 9: Decomposed sequence diagram for transpiler for lexical



after the transpiling done the cocos2d-x C++ code is converted to cocos2d-JS , below is the screenshot of the code conversion.

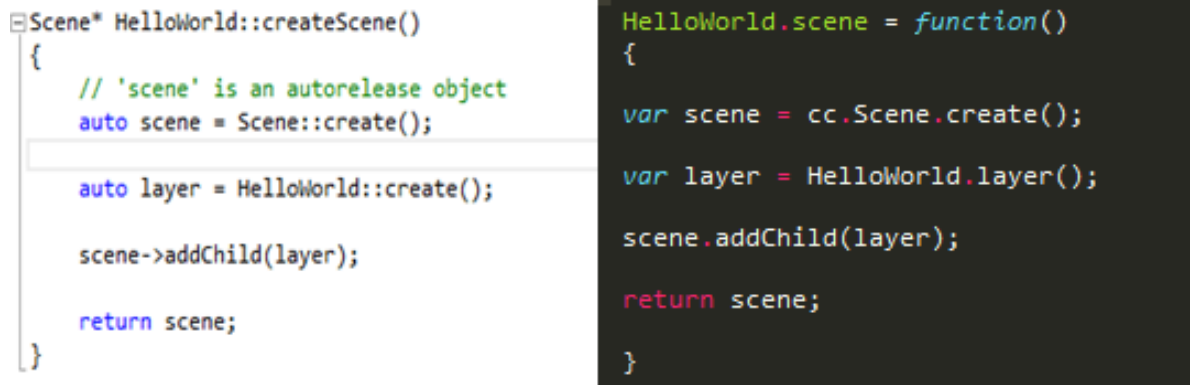
when the user select choice 1 then it takes the data from the externa trainingl file and then train the system via the training algorithm , when the user choose 2 the it it ask for the location of the file once the full path of the file is entered then it start the translation process and write the data in file the same location , by choice 3 it gets the data from the exisiting test file in the data folder and then translate it generally used for testing purpose.

Figure 10: Decomposed sequence diagram for transpiler for grammar



User have t select .

Figure 11: Transcompiled code from cocos2d C++ to cocos2d JS



The image shows two side-by-side code snippets. The left snippet is C++ code for a function named `createScene()` in the `HelloWorld` namespace. It creates a `Scene` object, creates a `Layer` object, adds the `Layer` to the `Scene`, and returns the `Scene`. The right snippet is the transcompiled JavaScript code. It defines a `scene` property on the `HelloWorld` object as a function that performs the same steps: creating a `cc.Scene`, creating a `HelloWorld.layer()`, adding it to the scene, and returning the scene.

```
Scene* HelloWorld::createScene()
{
    // 'scene' is an autorelease object
    auto scene = Scene::create();

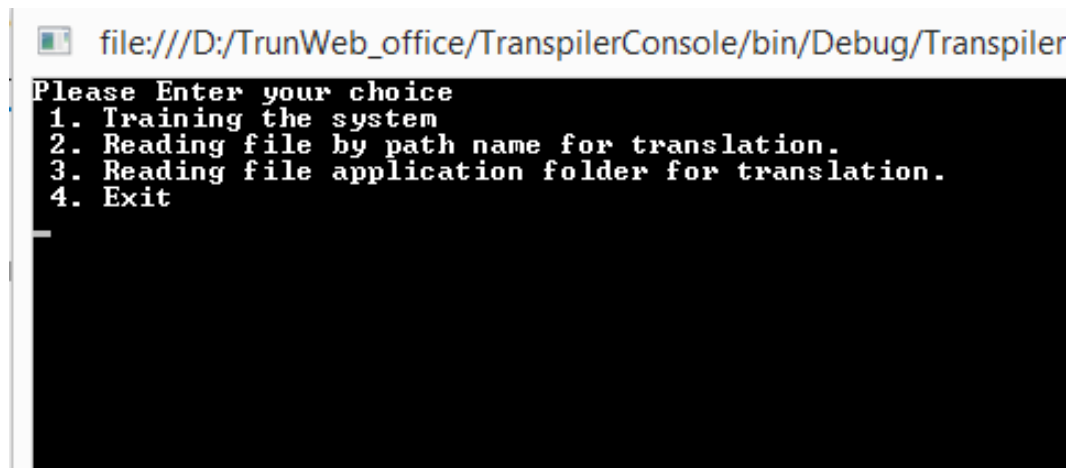
    auto layer = HelloWorld::create();

    scene->addChild(layer);

    return scene;
}

HelloWorld.scene = function()
{
    var scene = cc.Scene.create();
    var layer = HelloWorld.layer();
    scene.addChild(layer);
    return scene;
}
```

Figure 12: screenshot of the menu driven transpiler console



The image is a screenshot of a terminal window. The title bar shows the file path: `file:///D:/TrunWeb_office/TranspilerConsole/bin/Debug/Transpiler`. The terminal content displays a menu with the prompt "Please Enter your choice" followed by four numbered options: 1. Training the system, 2. Reading file by path name for translation, 3. Reading file application folder for translation, and 4. Exit. A cursor is visible on the line following the menu.

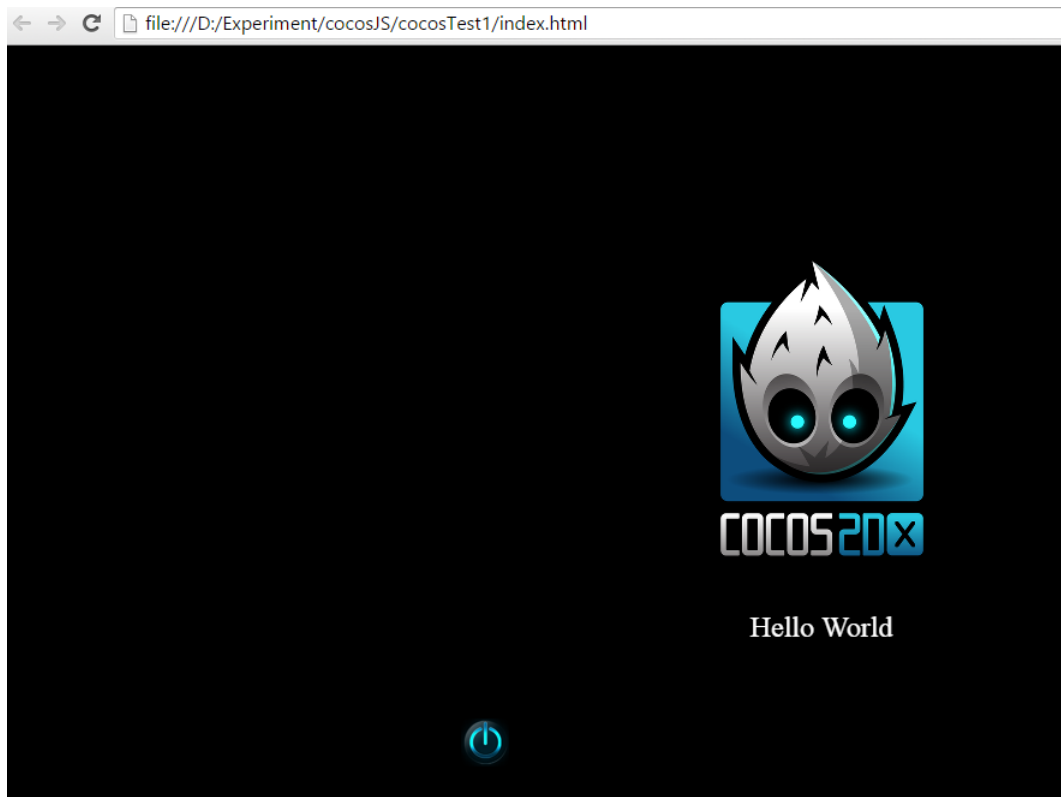
```
file:///D:/TrunWeb_office/TranspilerConsole/bin/Debug/Transpiler
Please Enter your choice
1. Training the system
2. Reading file by path name for translation.
3. Reading file application folder for translation.
4. Exit
```

7 Conclusion

7.1 Final result

The main objective of these thesis is to port the cocos2d C++ application to web browser as we can see from the result below it successfully converted to the web application using cocos2d JS, for that the transpiler was used which helped in translating the cocos2d C++ source code.

Figure 13: Screenshot of the coco2d JS application on web browser after transpiling



7.2 Limitation and deviation from hypothesis

There was no deviation from the hypothesis it went according to what we planned. There are some limitations because the two have some syntactical difference which was already discussed above in section observation of code structure of cocos2d C++ and cocos2d JS this exceptions sometimes effect the accuracy of translation, therefore the accuracy of transpiling remains between 80-90 % these code changes needed to be done by hand and then add it to the special cases xml , these helps as when the code is compiled again it gets the correct syntax from special cases xml and gets compiled with 100 % accuracy.

7.3 Future work

Future work would be training the system with new data when a new version of the framework is released. Then transpiling the class of TrunAPP ,extending the grammar.

8 Appendix

Listing 6: Code for training the system

```
class Training
{
    double[,] states = new double[4, 4] { { 0, 0.5, 0, 1 }, { 0.7, 0.5, 0.7, 0.7 },
    { 0, 0.7, 0.5, 0.7 }, { 0, 0.8, 0, 0.8 } };
    double[,] param = new double[4, 2];

    string[] separator = { ";", "{", "}", "\r", "\n", "\r\n" };

    string[] operators = { " ", "+", "-", "*", "/", "%", "&", "(", ")", "[", "]",
        "|", "^", "!", "~", "&&", "||", ",",
        "++", "--", "<<", ">>", "==", "!=", "<", ">", "<=",
        ">=", "=", "+=", "-=", "*=", "/=", "%=", "&=", "|=",
        "^=", "<<=", ">>=", ".", "[]", "()", "?:", "=>", "??", ":", "}" };

    public string[] calculateParam(string name)
    {
        double Kno = (4 / name.Length) > 1 ? 0.1 : (4 / name.Length);
        double ono = (1 / name.Length) > 1 ? 0.1 : (1 / name.Length);
        double sno = (1 / name.Length) > 1 ? 0.1 : (1 / name.Length);
        double ino = (5 / name.Length) > 1 ? 0.1 : (5 / name.Length);

        bool oregB, sregB;

        if (Array.IndexOf(operators, name) > -1)
        {
            oregB = true;
        } else
        {
            oregB = false;
        }
    }
}
```

```

if (Array.IndexOf(separator , name) > -1)
{
    sregB = true;
} else
{
    sregB = false;
}

double kreg = Regex.Match(name, @"^[a-zA-Z]+$").Success ? 1 : 0;
double oreg = oregB ? 1 : 0;
double sreg = sregB ? 1 : 0;
double ireg = Regex.Match(name, @"^[a-zA-Z0-9::]+$").Success ? 1 : 0;

param[0,0]=Kno; param[0,1]=kreg; param[1,0] = ono; param[1,1] = oreg;
param[2,0] = sno; param[2,1] = sreg; param[3,0] = ino;
param[3,1] = ireg;
string[] ret = new string[4];
if ((Kno > 0.5) && (kreg > 0.5))
{
    ret[0] = "keyword";
    ret[1] = name;
    ret[2] = Convert.ToString((Kno + kreg) / 2);

    return ret;

}
else if ((ono > 0.5) && (oreg > 0.5))
{
    ret[0] = "operator";
    ret[1] = name;
    ret[2] = Convert.ToString((ono + oreg) / 2);

```

```

        return ret;

    }
    else if ((sno > 0.5) && (sreg > 0.5))
    {
        ret[0] = "seperator";
        ret[1] = name;
        ret[2] = Convert.ToString((sno + sreg) / 2);

        return ret;

    }

    else
    {
        return ret;
    }
}

```

Listing 7: Reading the tokens from the file

```

while ((line = file.ReadLine()) != null)
{
    DataTable symbolTable = new DataTable();
    symbolTable.Columns.Add("type", typeof(string));
    symbolTable.Columns.Add("name", typeof(string));
    TranspilerConsole.utility.Translation tr = new
        TranspilerConsole.utility.Translation();

    string str = "";
    string st = peg.pegGrammarstr(line);
    if (string.Compare(st, "") != 0)
    {
        str = st;
    }
}

```



```

    }
else
{
    try {
        while (line != "")
        {

            line = line.Trim(' ', '\t');
            string token = analyzer.GetNextLexicalAtom(ref line);
            if (token != null)
            {
                string[] tok = splitLineString(token);
                symbolTable.Rows.Add(tok[0].Trim(), tok[1].Trim());
                string c = tok[1].Substring(tok[1].Length - 1);
                if (string.IsNullOrEmpty(c))
                {
                    str = str + tran.getTranslatedData(tok[0].Trim(),
                                                         tok[1].Trim()) + " ";

                }
            }
            else
            {
                str = str + tran.getTranslatedData(tok[0].Trim()
                                                    , tok[1].Trim());
            }
        }
    }
}

```

References

- [1] <http://www.truncad.de/>
- [2] <https://raw.githubusercontent.com/kripken/emscripten/master/docs/paper.pdf>
- [3] <http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/34913.pdf>
- [4] Roger Engelbert "*Cocos2d-x by examples beginners guide*" Packt Publishing Ltd, April 2013.
http://ftp.factor.lg.ua/books/Cocos2d-X_by_Example_Beginner_s_Guide.pdf
- [5] <http://www.cocos2d-x.org/wiki/Cocos2d-JS>
- [6] <http://stackoverflow.com/tags/cocos2d-html5/info>
- [7] http://www.cocos2d-x.org/wiki/How_to_bind_C++_to_Javascript
- [8] <http://blogs.adobe.com/flascc/2013/01/18/porting-a-c-opengl-game-to-run-in-the-browser>
- [9] https://developer.xamarin.com/guides/cross-platform/game_development/cocossharp/
- [10] <http://adobe-flash.github.io/crossbridge/>
- [11] <http://leaningtech.com/cheerp/>
- [12] <http://searchservervirtualization.techtarget.com/definition/platform>
- [13] <https://en.wikipedia.org/wiki/Cross-platform>
- [14] <http://www.cocos.com/doc/tutorial/show?id=2678>
- [15] https://raw.githubusercontent.com/cocos2d/cocos2d-x/v3/docs/framework_architecture.jpg
- [16] Torben/Egidius Mogensen "*Basic of compiler design*" lulu.com, August 2010.
(http://www.diku.dk/~torbenm/Basics/basics_lulu2.pdf)
- [17] <http://vschart.com/compare/c-plus-plus/vs/javascript>
- [18] <http://www.differencebetween.info/difference-between-javascript-and-cplusplus>
- [19] <https://forums.xamarin.com/discussion/30701/cocossharp-project-templates-for-visual-studio>

- [20] <https://github.com/leaningtech/cheerp-meta/wiki/Cheerp-Tutorial>
- [21] http://www.cocos2d-x.org/wiki/Native_Client_Environment_Setup
- [22] <https://developer.chrome.com/native-client/images/nacl-pnacl-component-diagram.png>
- [23] <http://www.gamefromscratch.com/post/2012/06/04/Cocos2D-HTML5-tutorial-1-Getting-set-up-and-running.aspx>
- [24] https://kripken.github.io/emscripten-site/docs/getting_started/downloads.html
- [25] https://kripken.github.io/emscripten-site/docs/getting_started_with_emscripten_and_vs2010.html
- [26] http://www.cocos2d-x.org/wiki/Emscripten_usage
- [27] <http://rengelbert.com/blog/cocos2d-html5-the-syntax/>
- [28] <https://developer.chrome.com/native-client/faq>
- [29] https://kripken.github.io/emscripten-site/docs/porting/guidelines/portability_guidelines.html
- [30] <https://hpyblg.wordpress.com/2010/05/30/simulating-pointers-in-javascript>
- [31] <http://www.adobe.com/devnet-docs/flascc/docs/Reference.html>
- [32] <http://rengelbert.com/blog/cocos2d-html5-the-syntax/>
- [33] http://www.adobe.com/devnet/flashplayer/articles/secure_swf_apps.html
- [34] Tobias widlund *Application programming interface for native web applications* . , Lulea university of Technology
- [35] Alon Zakai *Emscripten: An LLVM-to-Javascript compiler* . ,Mozilla
- [36] <http://stackoverflow.com/questions/10231868/pointers-in-javascript>
- [37] <https://github.com/alehander42/pseudo-python>
- [38] <http://www.cocos2d-x.org/wiki/Cocos2d-JS>
- [39] <http://blogs.adobe.com/flascc/>
- [40] <http://codereview.stackexchange.com/questions/113418/lexer-for-c-source-code>
- [41] <http://www.cocos2d-x.org/wiki/>
- [42] <https://www.draw.io/>

- [43] <http://logicpool.com/archives/30>
- [44] <http://discuss.cocos2d-x.org/t/performance-tests-and-comparison-of-cocos2d-x-3-2-and-cocos2d-js-3-0/16087>
- [45] <http://asmjs.org/faq.html>
- [46] https://en.wikipedia.org/wiki/Google_Native_Client
- [47] http://www.adobe.com/devnet/flashplayer/articles/secure_swf_apps.html
- [48] <http://bacsoftwareconsulting.com/blog/index.php/web-development/flash-website-advantages-and-disadvantages>
- [49] <http://mrle.ph/blog/2013/03/28/why-asmjs-bothers-me.html>
- [50] <https://gist.github.com/evanw/8258541>
- [51] <http://www.veracode.com/security/javascript-security>
- [52] https://kripken.github.io/emscripten-site/docs/porting/guidelines/portability_guidelines.htm
- [53] <http://leaningtech.com/cheerp/blog/2015/06/17/Cheerp-1.1>
- [54] <http://blog.commlabindia.com/elearning-development/flash-benefits-limitations>
- [55] <http://coffeescript.org>
- [56] David Chiang, *Hierarchical Phrase-Based Translation*, Information Sciences Institute, University of Southern California
<http://www.mitpressjournals.org/doi/pdf/10.1162/coli.2007.33.2.201>
- [57] Yusuke Oda, Hiroyuki Fudaba, Graham Neubig, Hideaki Hata, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura *Learning to Generate Pseudo-code from Source Code using Statistical Machine Translation* Nara Institute of Science and Technology, Japan, pp. 2-3
<http://www.phontron.com/paper/oda15ase.pdf>
- [58] <https://developer.chrome.com/native-client/faq>
- [59] https://kripken.github.io/mloc-emscripten_talk/cppcon.html/31