UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER SCIENCE

CS/IT Honours Final Paper 2019

Title: The Engineering of a Dance-Notation Visualization System Through the Iterative-Waterfall Approach

Author: Jordy Chetty

Project Abbreviation: DEDANCE

Supervisor(s): Dr Maria Keet

Category	*Min*	*Max*	Chosen
Requirement Analysis and Design	*0*	*20*	20
Theoretical Analysis	*0*	*25*	0
Experiment Design and Execution	*0*	*20*	0
System Development and Implementation	*0*	*20*	20
Results, Findings and Conclusion	*10*	*20*	10
Aim Formulation and Background Work	*10*	*15*	10
Quality of Paper Writing and Presentation	*10*		10
Quality of Deliverables	*10*		10
Overall General Project Evaluation (*this section allowed only with motivation letter from supervisor*)	*0*	*10*	
Total marks		**80**	

The Engineering of a Dance-Notation Visualization System Through the Iterative-Waterfall Approach

Jordy Chetty
University of Cape Town
Cape Town, Western Cape
chtjor001@myuct.ac.za

ABSTRACT

Most recent efforts relating to dance-notation animation systems generate static three-dimensional output derived from analysing the structure of an annotation on a best-effort basis. These annotations are derived from complex, paper-based notation schemes, making the systems difficult to use without prior knowledge of a dance notation scheme. They provide little value for dance learners in terms of being able to understand how dance moves are performed accurately due to information loss that occurs during conversion from notation to animation. Fundamentally, these systems are not able to produce meaningful output that can be utilized by dancers effectively. Using the Salsa dance style, we present a new dance-notation visualization system that both simplifies the annotation analysis step and eliminates information loss through the use of motion capture data rather than syntactical analysis. Additionally we add an interpolation component to blend separate segments of motion data. As a result, we improve the state of the art through the mitigation of information loss, removing the complexity associated with recording sequences of dance moves and through additional functionality.

CCS CONCEPTS

• **Software and its engineering → Requirements analysis**; **Software design engineering**; **Software implementation planning**; *Software development techniques*; System description languages.

KEYWORDS

motion data, software engineering, computer graphics, interpolation

1 INTRODUCTION

Conventional learning patterns for dance typically involve two parties, one that possesses competent knowledge about how movements are performed, other movements that can be performed in succession, and how they integrate within a rhythm count. Informally, this may be considered as the definition of the dance. The second party, the student, learns through observation or practice, or a combination thereof. Once adequate knowledge has been transferred to the student, dance movements can be easily identified, but conveying this information symbolically remains a challenging problem. The issue of encoding a dance style is challenged by variability in execution that can occur for a given motion and the data that is required to accurately denote how to perform a movement without losing information. The best efforts to achieve this store the position and orientation of vital limb information as time progresses along with a set of rules detailing where the movement can be performed. The set of data that is needed to convey this information is infeasible to write manually and confusing for a learner to understand. As a result, no standardized notation exists to accurately encompass the full set of dance movements for all dance styles. Cillekens [11] states that this possibly could be attributed to the complexity of recording time-series three-dimensional data. Cumulative error due to the unreliability of human recollection over multiple generations may cause the contemporary structure of a dance style to vary from its original form. Entire movements may also be lost within a single generation. This is influential as dance styles have cultural significance. As movements are forgotten, heritage is lost. As a result, systems have been created to expedite the notation process and visualize notations through animations, but are greatly constrained making them impractical for general use. Several notations exist for notating dance movements but Labanotation [12] has become most widely accepted, and is thus used in most dance-notation animation systems. Other notation schemes are either more complex or not descriptive enough to be able to record dance movement sufficiently. The use of Labanotation in these types of systems is debatable as they only implement a subset of it due to its inherent complexity and lexis. Fundamentally the expressiveness of existing systems is dependent on the notation that is used to represent movement. On the other hand, the expressiveness of notation schemes increases as the complexity of the notation increases. This makes designing effective systems of this nature a difficult challenge. In this paper, we present Salsational, a system for visualizing dance-notations through the use of motion-captured data. In contrast to prior systems of a similar nature, our system can display three-dimensional movement sequences as opposed to static images and does not require technical knowledge of a dance style or notation system to store and record movement. It functions by using a global or local movement data store that users can contribute to in a simple way, which can then be used to create sequences of movement. The requirements for the system were specified by Angus Prince of Evolution Dance Studio (EDC), and can be found in section 4.1. Mr. Prince helped to test and evaluate the system throughout the development life cycle.

This system was created in conjunction with a complementary system, created by Micara Marajh and Alka Baijnath, which allows Description files (referred to in appendix A) to be loaded into the system. The scope of this paper is restricted to the graphical aspect of the Salsational system, but references are made to the description module for completeness. The intellectual property rights of the source code is held and maintained by the creator, Jordy Chetty. The source code for this system is released as open-source software to the public and licensed under the GNU General Public License (GPL) V3. Although the software is open-sourced, the system contains both licensed and unlicensed code from several vendors. The paper is structured as follows. In section 2, we discuss related work in dance-notation visualization systems and provide background theory on subjects referred to in this paper. In section 3, we discuss the software development methodology used to create the system. In section 4, we discuss the requirement analysis phase of development. In section 5, we discuss the system development and implementation phase. In section 6, we discuss the results and provide a discussion of the development process. Finally, we state conclusions in section 7.

2 BACKGROUND

This section first describes the state of the art in dance-notation visualization systems. Then background theory is provided for motion capture formats and auxiliary processes. Interpolation and approximation methods are then discussed briefly.

2.1 Existing Dance-Notation Visualization Systems

LabanWriter is a notation-transcribing application which allows users to graphically create scores by way of a graphical interface. The files generated by this application were used by LabanDancer [13] to convert scores into graphical representations. The symbols in LabanWriter files were transformed and placed into a data structure which was then used as a reference to drive the animation of a three-dimensional model. The movement of the model was synthesised using IKAN, an inverse kinematics algorithm. The use of inverse kinematics as opposed to animation data presents possible inaccuracies in the performance of the animation as it is not data-driven. Furthermore, the limitations of LabanWriter directly affect the results of [13]. Laban Editor [7] has functionality for both creating scores and generating animations. Similarly to LabanWriter, scores can be written using an graphical interface. The application stores the movement data in Labanotation Data (LND) format. Animation is performed by converting LND using motion conversion template files. These files generate the data required to drive the movement of a figure. In contrast to [13], the system offers several conversion methods which enable the figure to perform the animation differently by using different templates. [14] stated this as problematic as the quality of the animations relied on the quality of the template files available. Both [13] and [7] only allow a single figure to be animated at a time. Life Forms, a proprietary software platform, addressed the issue of single character animation by allowing multiple figures to be animated using motion capture data. The system can import and export several popular motion capture file formats. It is packaged with additional motion capture data that can be used to extend animation sequences by using its blending functionality. Users can also edit models graphically, and create new poses from existing poses by editing the skeletal structures imported into the program. It can be integrated with popular applications such as Maya. It is more extensible and flexible than the applications previously described, but the support for Labanotation is limited.

2.2 Motion Capture

The BioVision Hierarchy (BVH) file specification, shown in Figure 11 is used to encode data recorded using motion capture systems. The file consists of two sections. The first section describes a skeleton hierarchy and segment information. The hierarchical description implies that parent segment transformations affect all child transformation segments. Each segment contains a channel description, which parameterizes segment animation and defines the order in which transformation are composed. Each channel description identifies the axes along which a segment can be rotated and translated. The second section contains motion samples specified in Euler angles. Each motion sample is defined on a separate line. The data for each channel is listed in the order in which the channel appears in the hierarchy description on every line [9]. Segmentation of motion capture is the process of isolating a sequence of motion data, usually for the purpose of capturing a recognizable movement. In practice, motion capture systems record continuous motion and thereafter the data is segmented manually or via an automated process.

2.3 Interpolation

n-dimensional Bezier curves [3], defined as

$$B(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i P_i$$

with $t \in [0,1]$, points $P_0, P_1, ..., P_n$, and

$$\binom{n}{i} = \frac{n!}{i!(n-i)!}$$

are used interpolate and approximate curves between a set of n control points. Linear interpolation between two points can be achieved by using a linear Bezier curve, and approximating curves for higher dimensions.

A spline is a piece-wise function defined by polynomial functions. They are advantageous in comparison to other forms of interpolation as they . Multiple spline functions exist, with common variants being the Hermite Spline function, the Catmull-Rom spline function [2], being used in computer graphics due to their computational efficiency and simplistic nature. In comparison to Bezier curves which only pass through the first and last control points when more than two control points are specified, spline curves can pass through each specified control point - called interpolating - or near to each control point - approximating - depending on the parameterization of the function.

Euler angles [5], generally denoted by

$$(\alpha, \beta, \gamma)$$

are three angle that, in combination, define an axis and a rotation value. They are used to represent the orientation of a rigid body with respect to a fixed coordinate system within a space.

Quaternions [6], given by

$$a + bi + cj + dk$$

can be used to represent rotations and orientation in space. In contrast to Euler angles, they are not subject to gimbal lock - which

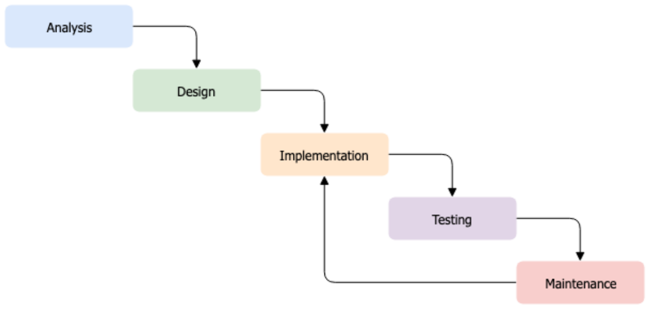

Figure 1: The Iterative-Waterfall Model, displaying the feedback loops that were used to create iterative behaviour.

results in the loss of a degree of freedom, and are more computationally efficient.

3 SOFTWARE DEVELOPMENT METHODOLOGY

The project was developed using a hybrid-model of the Iterative and Waterfall approaches, the Iterative Waterfall [4], pictured in Figure 1. This model provided a degree of flexibility in the development process by way of the Iterative process, but simultaneously enabled rigidity through the Waterfall process. The implication of this approach allowed the client to review the artefacts of each iteration and to provide modifications to the original specification, but with initial requirements provided during the requirements gathering phase remaining largely intact. This provided equilibrium in the development life cycle due to limited client interaction having been expected as opposed to an active-engagement model common to Agile approaches. With respect to the software creation process, this allowed multiple build cycles to run concurrently. Development iterations comprised implementation, testing, and maintenance. The implementation phase initiated the actual programming of a set of tasks. The testing phase assessed the intended functionality of the system through a variety of testing techniques, and the maintenance phase was used to enhance and revise the functionality of the outputs of each iteration. Three iterations were conducted during the development period. Iterations one and two were evaluated by members of EDC. The iterative cycles that were conducted are discussed in sections 5.6, 5.7 and 5.8.

4 REQUIREMENT ANALYSIS

The requirement analysis phase was concerned with the conceptual design of the system with respect to the user requirements. The first section briefly describes the requirement-gathering phase, which is then followed by a description of stakeholders and the architectural design model used to analysis the requirements.

4.1 Requirement Elicitation

Requirements were collected from Angus Prince during the requirement-gathering phase of development. A meeting took place to discuss the the core functionality of the system. The requirements describe the functional and non-functional aspects of the system needed to aid the teaching process.

(1) Motion playback: Movements should be displayable to the end-user. In particular, the user should be able to view both the leader and the follower performing a sequence.
(2) Multi-angle camera: The movements of the figure should be viewable from multiple angles in order for vital limbs to be seen without occlusion as well as to complement the learning process. A top view and side view were specified explicitly.
(3) Graphical user interface: The system should be navigable using a graphical interface that the end-user can easily manipulate in order to use the application functionality.
(4) Line of dance: The line of dance should be able to be viewed by the end-user while a movement is being played. This is to allow the end-user to understand the how the figures should move relative to the space available to them.
(5) Multi-colored figures: The animated figures should be displayable in different colors in order for the end-user to be able to distinguish between the leader figure and the follower figure during movement playback.

4.2 Stakeholders

The stakeholders in the system were identified as the users of the system, developers of the system and maintainers of the system. The users are further defined as Salsa instructors.

4.3 4+1 Architecture View Model

The 4+1 Architecture View Model [8] was used to create a high-level abstraction of the system derived from the requirement analysis phase. This architectural framework was used to separate and identify concerns of the system of different stakeholders that were used to inform design decisions during the iterative phase of development. The scope of the architecture is limited to the Salsational system and and user interface. The following sections address each of the viewpoints in the model and primary concerns of each of the stakeholders that they represent. The resulting software design can be seen in Figure 9.

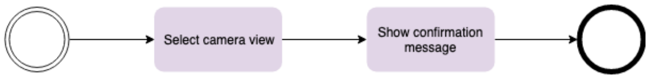

Figure 2: State diagram for changing the camera

4.3.1 *Logical View.*

The logical view aims to identify the manner in which users interact with the system. Each interaction was modeled using state diagrams to show the behaviour of the system in

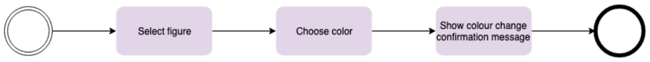

Figure 3: State diagram for changing the figure color.

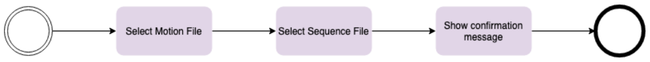

Figure 4: State diagram for description file loading

response to user-provided input. The provided figures address how the system is achieves the functional requirements of the system.

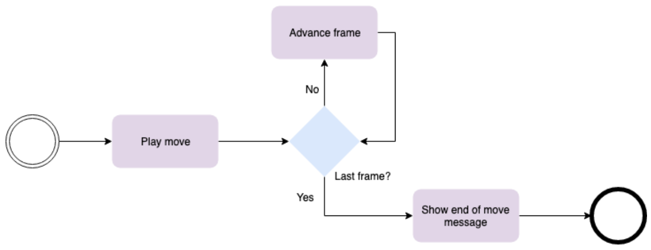

Figure 5: State diagram for playing a move

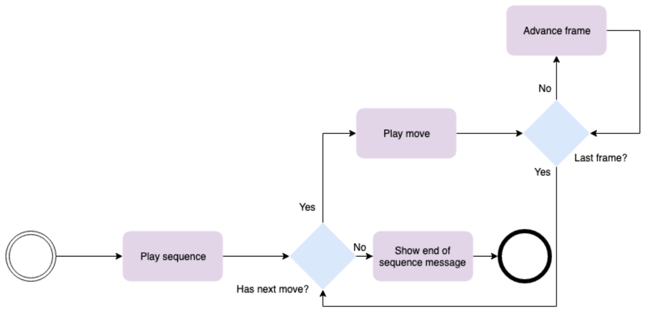

Figure 6: State diagram for playing a sequence

4.3.2 Development View. The development view shows the high-level software design of the system which can be seen in Figure 7. The viewpoint is depicted using a package diagram and shows the interactions between modules. The model addresses the issues of maintainability and structure of the system. It shows the separation of components into a layered architecture, as well as the component-wise decomposition of functionality into cohesive units. The diagram additionally displays the core services provided by the system in order to meet the functional and non-functional requirements as specified by the requirements.

4.3.3 Process View. The process view describes the interactions between components in the system. It was created using an activity diagram and is shown in Appendix B.1. It shows the basic engagement between users and the system in which a user loads description files and then plays a series of sequences and moves until completion.

4.3.4 Physical View. The physical viewpoint conveys the deployment of the system within the user environment and is shown in Figure 8. This view is intended to model the conditions under which the system is expected to function.

4.3.5 Scenarios. Scenarios were created from user stories that were modeled from the gathered requirements, which can be seen in Table 1. Each user story was given condition(s) of satisfaction as evaluation criteria. The scenarios can be used to illustrate the functioning of the elements of the architect described above.

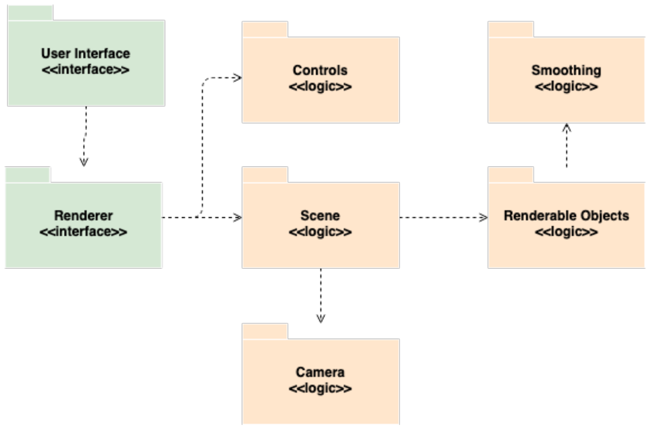

Figure 7: Package diagram exhibiting the high-level structure of component interactions in the system, as well as the separation of components into functional layers.

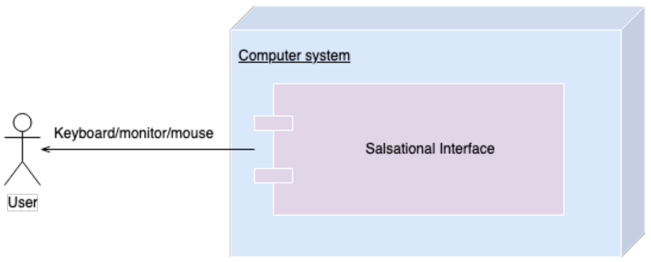

Figure 8: Deployment diagram showing the physical environment of the system.

5 SYSTEM DEVELOPMENT AND IMPLEMENTATION

This section describes the manner in which artefact specifications were realised and the implementation of software engineering processes that were used during the project life cycle.

5.1 Programming Language

C++11 was used to program the system. Despite bindings of OpenGL to other languages, support for external libraries that were needed during development were limited. Version 11 introduced several features which were used in the system. The list below describes core features of the version 11 specification and their significance to the system:

(1) Type inference through the auto data type
(2) The override identifier for specifying overridden virtual functions in child classes
(3) Default initialization of in-class values
(4) std::chrono for tracking time during frame-based animation playback
(5) Operator overloading

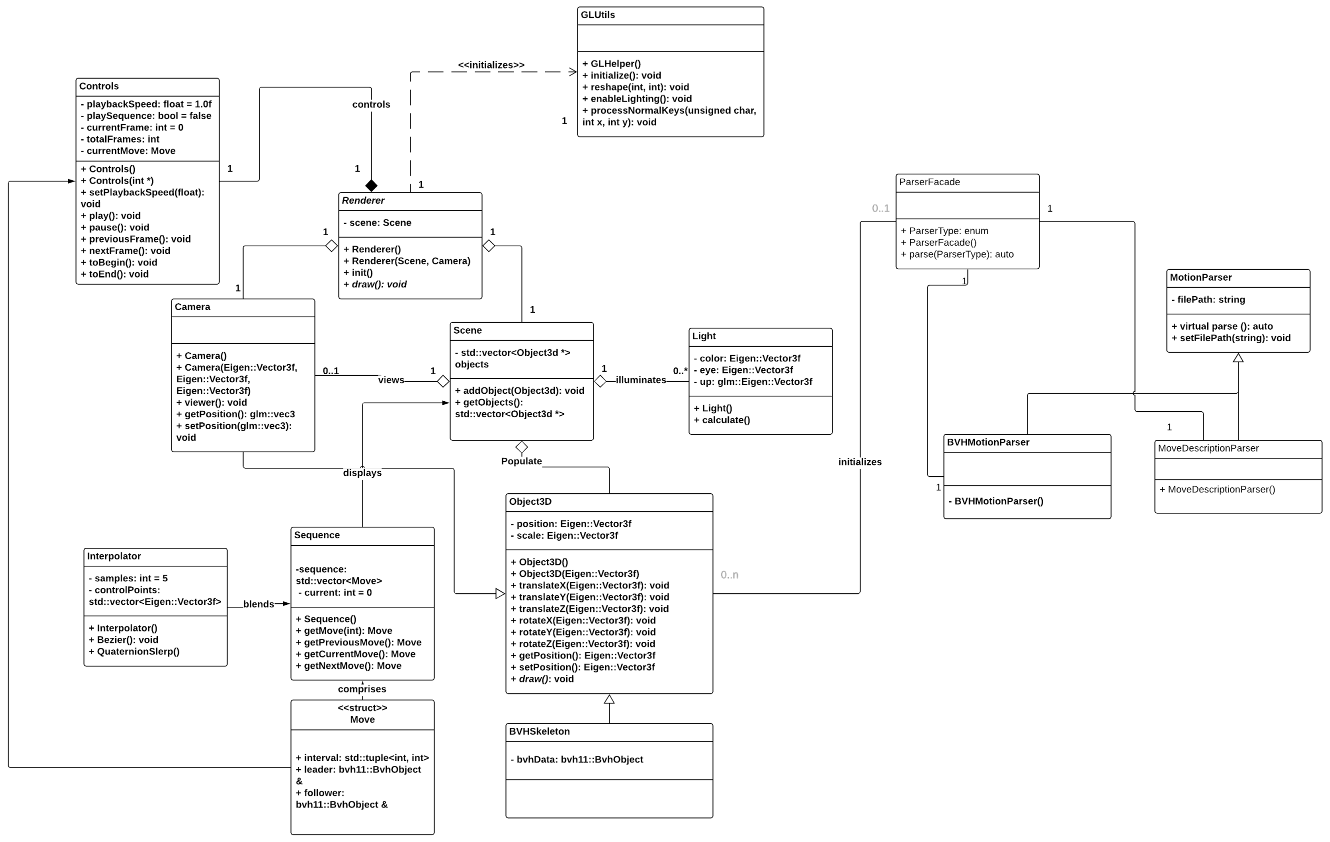
GLUtils
+ GLHelper()
+ initialize(): void
+ reshape(int, int): void
+ enableLighting(): void
+ processNormalKeys(unsigned char, int x, int y): void
<<initializes>>
Controls
- playbackSpeed: float = 1.0f
- playSequence: bool = false
- currentFrame: int = 0
- totalFrames: int
- currentMove: Move
+ Controls()
+ Controls(int *)
+ setPlaybackSpeed(float): void
+ play(): void
+ pause(): void
+ previousFrame(): void
+ nextFrame(): void
+ toBegin(): void
+ toEnd(): void
controls
Renderer
- scene: Scene
+ Renderer()
+ Renderer(Scene, Camera)
+ init()
+ draw(): void
Camera
+ Camera()
+ Camera(Eigen::Vector3f, Eigen::Vector3f, Eigen::Vector3f)
+ viewer(): void
+ getPosition(): glm::vec3
+ setPosition(glm::vec3): void
views
Scene
- std::vector<Object3d *> objects
+ addObject(Object3d): void
+ getObjects(): std::vector<Object3d *>
illuminates
Light
- color: Eigen::Vector3f
- eye: Eigen::Vector3f
- up: glm::Eigen::Vector3f
+ Light()
+ calculate()
displays
Populate
ParserFacade
+ ParserType: enum
+ ParserFacade()
+ parse(ParserType): auto
MotionParser
- filePath: string
+ virtual parse (): auto
+ setFilePath(string): void
BVHMotionParser
- BVHMotionParser()
MoveDescriptionParser
+ MoveDescriptionParser()
initializes
Object3D
- position: Eigen::Vector3f
- scale: Eigen::Vector3f
+ Object3D()
+ Object3D(Eigen::Vector3f)
+ translateX(Eigen::Vector3f): void
+ translateY(Eigen::Vector3f): void
+ translateZ(Eigen::Vector3f): void
+ rotateX(Eigen::Vector3f): void
+ rotateY(Eigen::Vector3f): void
+ rotateZ(Eigen::Vector3f): void
+ getPosition(): Eigen::Vector3f
+ setPosition(): Eigen::Vector3f
+ draw(): void
Interpolator
- samples: int = 5
- controlPoints: std::vector<Eigen::Vector3f>
+ Interpolator()
+ Bezier(): void
+ QuaternionSlerp()
blends
Sequence
-sequence: std::vector<Move>
- current: int = 0
+ Sequence()
+ getMove(int): Move
+ getPreviousMove(): Move
+ getCurrentMove(): Move
+ getNextMove(): Move
comprises
<<struct>>
Move
+ interval: std::tuple<int, int>
+ leader: bvh11::BvhObject &
+ follower: bvh11::BvhObject &
BVHSkeleton
- bvhData: bvh11::BvhObject
1
0..1
0..*
0..n

Figure 9: Salsational Class Diagram

	User Story	Condition(s) of Satisfaction
1	As a user, I want to view the leader and follower from different angles, so that I can clearly see how certain movements are performed.	The figures can be viewed from several viewpoints that clearly show different limb positions.
2	As a user, I want to see the leader and follower performing different movements, so that I can understand how moves are performed.	The user does not have to reinitialise the program in order to view multiple movements.
3	As a user, I want to be able to interact with the system using an interface, so that I can use the system easily.	The user can use valid system functions through a user interface.
4	As a user, I want the leader and follower to be drawn in different colours, so that I can distinguish between them.	The user can clearly distinguish between the leader and follower when a movement is played.
5	As a user, I want to see the line of dance, so that I know where the leader and follower should be standing.	During a movement playback, the line of dance is drawn on screen.

Table 1: User stories and conditions of satisfaction captured during the requirements gathering phase and subsequent meetings after consultation with the client

5.2 Frameworks and Libraries

OpenGL 2.1 was used to program the rendering engine, using FreeGLUT for window configuration and I/O processing, and GLU for utility functions. FreeGLUT was chosen to compensate for the limitations of the original GLUT library, predominantly the implementation of the main loop function. The use of legacy OpenGL over modern OpenGL was to improve support for user systems and as performance was not an intended system design goal. The system currently has support for loading and displaying BVH files. This was chosen as it is most widely adopted, and most animation systems support importing and exporting of this format. The Dear ImGui C++ library was used to create the user interface. The Catch testing library was used to perform unit testing on system components.

5.3 Data Structures

5.3.1 Description files. Description files comprise Move and Sequence file pairs and represent movement-motion mappings for a given dance style. The are the primarily input into the system.

5.3.2 Mathematical Types. The structures required for the system predominantly comprised mathematical types, and operations on those types. For this purpose, Eigen was used to provide vector, matrix, quaternion and Euler angle types. It provided *n*-ary vector objects for storing renderable object data and matrix types for performing transformations. Quaternion and Euler angle types were used for processing during smoothing calculations. The library was chosen because it multi-platform, does not require to be separately installed by users, and for its minimal storage footprint.

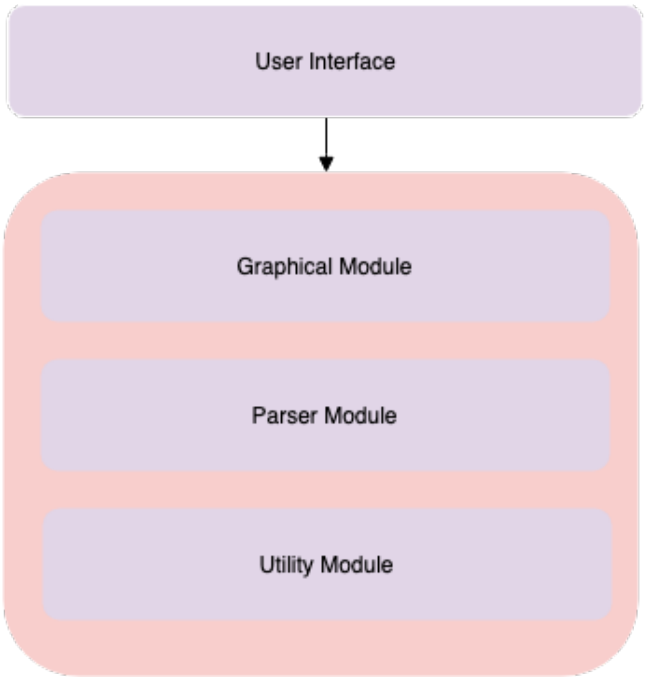

Figure 10: Salsational architecture diagram, showing the high-level structure of the system consisting of conceptually-divided layers to separate the front-end and logic.

5.4 Third-Party Services and Processes

CMake, Trello, Git, GitHub, Travis.CI, CodeFactor, Instruments, and Doxygen were used within the development life cycle. The project source code was built using the CMake build system. A kanban board was created on the Trello and platform extensions on the platform were used to control the software creation life cycle, and manage sub-tasks. This enabled work flow management, constraint identification and relationship establishment between tasks. Where a unit of work was actively being worked on, the task was moved to an active board, and to a completed board when finished. Git was used as the primary version control system. GitHub was used

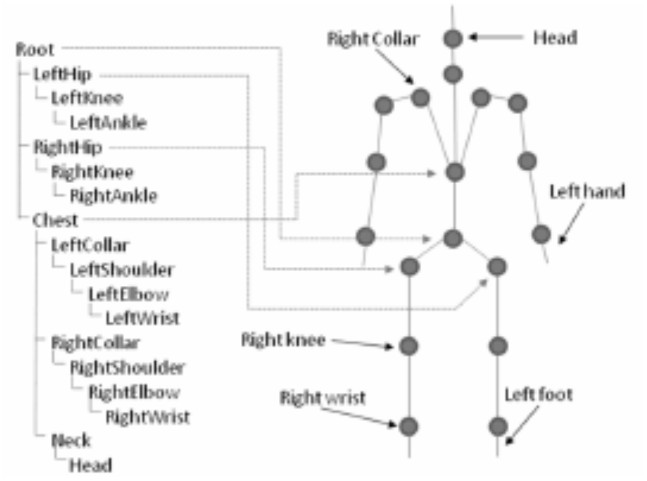

Figure 11: A visualization of the BioVision Hierarchy File skeleton hierarchy, showing the skeletal structure for several segments. [A Method for Comparing Human Postures from Motion Capture Data. Wei-Yang Tang et al. 2010.]

for remote code storage. Commits were performed irregularly. Initially, the project scaffold was generated entirely from the master branch. Thereafter, new branches were created. In this way, the master branch always contained a stable version of the software. Conceptually, the branches were categorised according to their primary functions, and suffixed with the category. A utility script, update_branches.sh, was used to synchronize every branch with the master branch and to push changes to the remote repository. Travis CI was used for continuous integration throughout the project for build notifications and error logging. Builds were initiated upon every push to the remote server. In this instance, GitHub. In the event of build errors, automated emails were sent. The build matrix included build operations across several operating systems, and was tested for Linux and Mac compiled with g++ using Make. Automated code review was conducted using CodeFactor. Each commit to the remote master branch was cloned to CodeFactor and issues raised within their online interface. The CodeFactor system ranked each class using a scale of A-F. The cumulative scores of each class determined the overall rating of the code. The code format was assessed using the Google style guide. The system contains GitHub Flavored Markdown (GFM) [1] which contains descriptions of the system as well as sample code. Additionally, Doxygen-style documentation is provided. Executable builds were profiled intermittently using Instruments, a built-in MacOS profiling tool. Where obvious performance issues occurred, the stack trace was observed to identify bottlenecks in the system and revisions were considered and made to improve the system running time. Throughout the development emphasis was placed on producing working software before refactoring tasks were completed, this was generally performed on completion of a functional class or complex method.

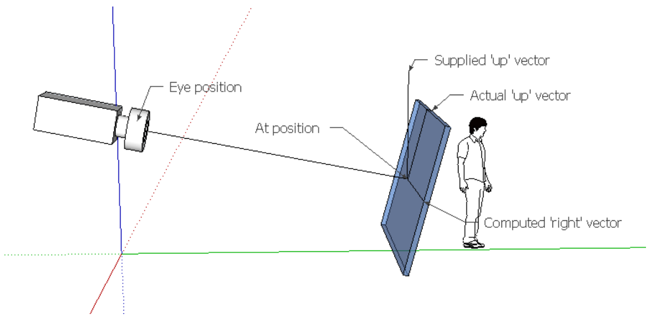

Figure 12: A visualization of the effects of the arguments to the *gluLookAt* function.

5.5 Software Quality Characteristics

The ISO/IEC 25010:2011 Standard addresses issues regarding software quality requirements and evaluation within the software development life cycle. The standard defines the Product Quality Model which characterizes the calibre of a software product using eight attributes, each of which comprise several sub-characteristics for evaluation. Of the eight, three were used that were relevant to the context of the software. This was to ensure that the system had been produced using modern techniques and in a robust manner.

5.5.1 Maintainability. Maintainability was achieved through the use of modular components which interface with one another uniformly. Provided that the interface between two interacting components remains unchanged, parts of the system can be interchanged with one another. A consequence of the modular design is modifiability of the components themselves. The division of components into functional subsystems creates functional cohesion model in the system. Functionality is divided in terms of the user requirements outlined in table 1.

5.5.2 Portability. Portability was achieved by using cross-platform external libraries and standard library headers exclusively. External dependencies can be installed using package managers on Unix-based systems. OpenGL 2.1 was used to facilitate wider system support over OpenGL 3.x. Installation and installation scripts can be automatically generated through the use of CMake.

5.5.3 Usability. Usability considerations were significant in the development of client application due to the computer-related competency of the end-users. Nielsen's 10 Usability Heuristics for User Interface Design were used to guide the design of the graphical interface to facilitate learnability of the system towards users. This was reflected in the design of the interface components. Error recovery was achieved by performing internal validation on user actions where input was required.

5.6 Iteration 1: Rendering System and Line of Dance

5.6.1 Implementation. The data for the motion data rendering process was acquired freely from the Carnegie Mellon University (CMU) Motion Database. Fifteen files were used, each of which contained several distinguishable movements. Segmentation was performed internally on each file in order to identify and name each dance move. The segments were then verified with Angus Prince during a meeting for accuracy and metadata. The accuracy validation involved resolving whether segmented sequences fairly represented each dance move. The metadata for each segment comprised the name of the dance move, which were recorded on a best-effort basis prior to the meeting and corrected during the consultation. A third-party library, BVH11, was used to parse and render each file between a subset of frames. The line of dance was implemented as a standard grid across the viewable area and drawn iteratively using the OpenGL line primitive.

5.6.2 Testing. Unit testing against the output of the BVH11 transformation method was performed. The global and local transformation of a single segment within a sample BVH file was manually calculated and tested against the library output. Further, acceptance testing for graphical output was conducted at EDC with Angus Prince and one EDC student. Minor revisions were proposed in order to improve the visibility of dancing figures.

5.6.3 Maintenance. Perfective maintenance was conducted to increase the radius of the limbs in each figure in order to improve visibility of the figures. The implementation of the coloring of each of the figures was modified enabling them to be dynamically changed during run-time.

5.7 Iteration 2: Camera and User Interface

5.7.1 Implementation. The camera component was designed in collaboration with EDC. Prior to implementation, several high-level configurations for the camera were given by Angus Prince, on the premise that the specified viewpoints assisted with the learnability of moves. Using the GLU toolkit, a camera is parameterized using three vectors in order to specify a viewing transform, which can be seen in Figure 12. Full control of the camera system was presented during an evaluation session at EDC. The user interface was constructed in C++ using the ImGui framework. Separate interface panels were used with the purpose of dividing disparate units of functionality. The viewpoints are shown in Figures 14, 15 and 16.

5.7.2 Testing. The outputs of the iteration were discussed at meeting at EDC. Angus Prince, one Salsa instructor, and three learners were present. The preset camera views, camera control system and user interface were discussed. From the feedback that was given it was found that the mechanism to control the camera system was not intuitive to use as well as the use of sliders. Improvements to the camera system were to provide additional pre-configured views in the form of an upper- and lower-body view, the use of a mouse to control the camera over the floating sliders. Improvements to the user interface were suggested to make the user aware of changes to the state of the system when actions were performed.

5.7.3 Maintenance. Adaptive and perfective maintenance was conducted on the system following the output from the testing phase to improve the system based on the suggestions that were given. The low-level camera controls were removed and replaced by pre-defined viewpoint buttons. The floating sliders were also removed. The additional viewpoints were also added to the system.

5.8 Iteration 3: Smoothing System

5.8.1 Implementation. Given two keyframes (a, b) from separate BVH motion file sources with the same skeleton layout, we aimed to create a smooth transition between the two frames dynamically. From [10], the position of a rigid body can be specified fully by the composition of rotations and translations, in which case we require two separate interpolation functions to generate intermediate samples. The rotation interpolation function would be applied to all BVH rotational channels and the translation interpolation function to all positional channels.

Bezier curves were used for positional interpolation and Quaternion Spherical Linear Interpolation (Slerp) for animating the rotation of each channel. A preliminary conversion from Euler Angle to Quaternion representation was required before interpolating rotational channels due to BVH storing data as Euler angles and trivially needed to be converted back to Euler Angles prior to rendering. The addition of additional control points was supported in order to refine the blending of the transitions between two files.

5.8.2 Testing. Unit testing for different parameterizations of the Bezier curve were performed with the value of t incremented in constant intervals of 0.2 per unit test on the same parameterization. Outputs were truncated to four decimal places. Similarly, the quaternion interpolation function was tested in constant intervals of 0.2 for $t \in [0, 1]$.

5.8.3 Maintenance. A final refactoring of the system was performed during this stage. The structure of the system was modified in order to conform to the class specification in Figure 9.

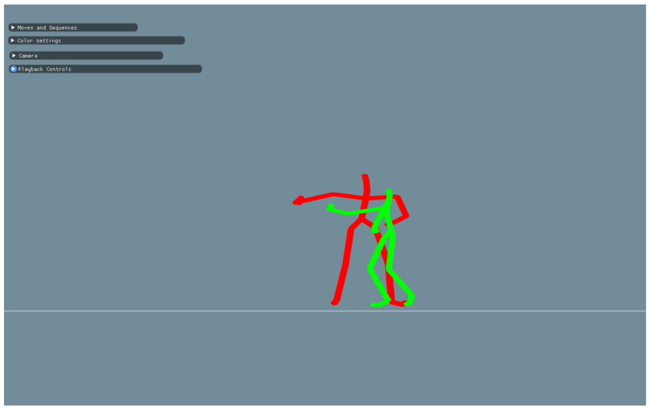

Figure 13: The Salsational System

6 RESULTS AND DISCUSSION

This system was intended to be an application of software engineering to the creation of a dance-notation visualization system that could be used within Salsa learning environment. The most important measure of the success of this system is in its ability to meet the requirements set out at the beginning of the development. Generally positive feedback was given during the meetings that were conducted with members of EDC which gives reason to believe that the system will be able to assist Salsa instructors in order to accomplish their goals. Unfortunately a final validation meeting was not able to be conducted to confirm the final changes expressed by EDC during the second iteration, although they were completed.

Aside from supporting Salsa, the system is designed to scale to support other pair-based dance styles. With novel modifications to the rendering process, support for arbitrary numbers of figures

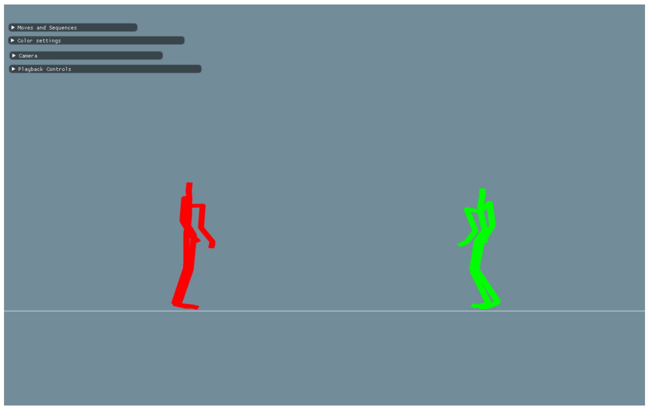

Figure 14: Salsational camera side view

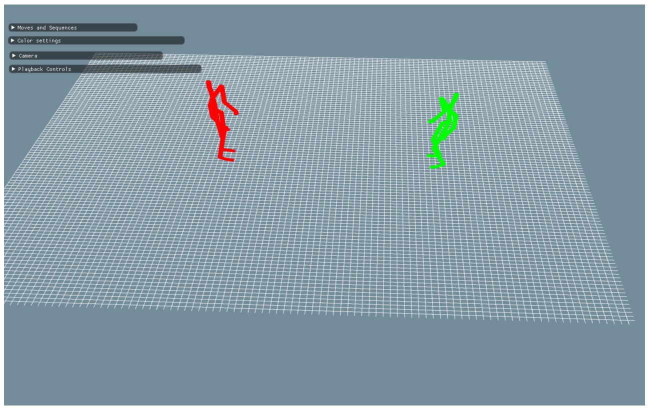

Figure 15: Salsational System top view

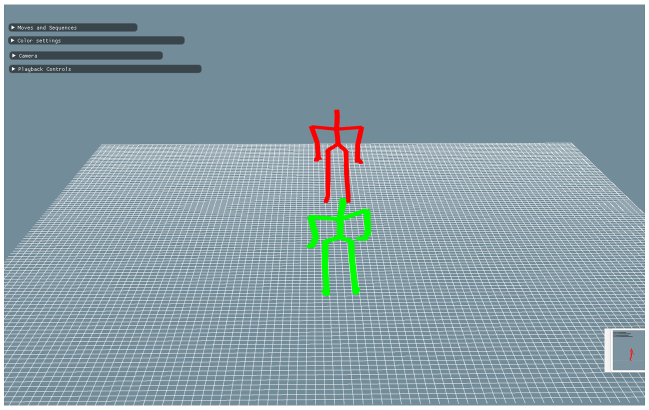

Figure 16: Salsational camera behind view

can be achieved to make the system extensible to a greater class of dance styles.

7 CONCLUSIONS

Salsational is designed to be a dance-notation visualization tool, with the primary goal to support the Salsa learning process using motion data. Salsational improves current dance-notation visualization systems through the use of motion data.

7.1 Future Work

Additions to the system include skinning and audio playback. A port of the system to Python, Unity or Maya would allow SMPL models to be used which would enable automatic deformable characters to be displayed in the system. This would be advantageous as different body types would be able to be viewed and adjusted to see how a performance would change depending on the physique of a learner. Support for additional motion data formats would be welcomed.

7.2 State of the Art System

The system improves on the state of the art by providing animated movement and dynamic sequence playback. The use of motion data as a means to express notation is something that has not been created before in this field.

7.3 A Dance Learning Tool

Using the above, we also believe that the system can be used by Salsa instructors in order to help teach learners dance movements in a simplified manner. We believe that the use of motion data will be effective within the learning process.

7.4 Smoothing

Salsational can create a contiguous motion sequences from separate motion files by using interpolation to generate dynamic motion between two source motion files. The accuracy of the dynamic motion is improved by the additional of intermediate controls points between the two files.

ACKNOWLEDGMENTS

I'd like to extend my sincerest gratitude towards our supervisor, Maria Keet, for her continued support and warming cheerfulness from start until end, and Angus Prince of Evolution Dance Company his invaluable insight, dedication and enthusiasm towards the Salsational project. Finally I would like to thank my parents for the opportunity and that they have given me and the sacrifices they have made for my sake to ensure that I am provided for, to the best of their abilities. I can confidently say that I would not be here today if it were not for both of you.

REFERENCES

[1] 2019. GitHub Flavored Markdown Spec. view-source:https://github.github.com/gfm/#what-is-github-flavored-markdown-

[2] Edwin Catmull and Raphael Rom. 1974. A CLASS OF LOCAL INTERPOLATING SPLINES. In *Computer Aided Geometric Design*, ROBERT E. BARNHILL and RICHARD F. RIESENFELD (Eds.). Academic Press, 317 – 326. https://doi.org/10.1016/B978-0-12-079050-0.50020-5

[3] Paul De Casteljau. 1959. Courbes à pôles. *National Industrial Property Institute (France)* (1959).

[4] Dr Nitin Deepak and Dr Shishir Kumar. 2015. Flexible Self-Managing Pipe-line Framework Reducing Development Risk to Improve Software Quality. *International Journal of Information Technology and Computer Science* 07 (06 2015), 35–47. https://doi.org/10.5815/ijitcs.2015.07.05

[5] Leonhard Euler. 1776. Novi Commentarii academiae scientiarum Petropolitanae. *Nr* 20 (1776), 189–207.

[6] William Rowan Hamilton. 1848. XI. On quaternions; or on a new system of imaginaries in algebra. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 33, 219 (1848), 58–60.

[7] K. Kojima, K. Hachimura, and M. Nakamura. 2002. LabanEditor: Graphical editor for dance notation. In *Proceedings. 11th IEEE International Workshop on Robot and Human Interactive Communication.* 59–64. https://doi.org/10.1109/ROMAN.2002.1045598

[8] P. B. Kruchten. 1995. The 4+1 View Model of architecture. *IEEE Software* 12, 6 (Nov 1995), 42–50. https://doi.org/10.1109/52.469759

[9] Maddock Meredith, Steve Maddock, et al. 2001. Motion capture file formats explained. *Department of Computer Science, University of Sheffield* 211 (2001), 241–244.

[10] Ken Shoemake. 1985. Animating rotation with quaternion curves. In *ACM SIGGRAPH computer graphics*, Vol. 19. ACM, 245–254.

[11] Colwyn Trevarthen and Stephen N. Malloch. 2000. The Dance of Well-being: Defining the Musical Therapeutic Effect. *Nordisk Tidsskrift for Musikkterapi* 9, 2 (2000), 3–17. https://doi.org/10.1080/08098130009477996 arXiv:https://doi.org/10.1080/08098130009477996

[12] Rudolf Von Laban. 1928. *Schrifttanz.* Universal-edition.

[13] Lars Wilke, Tom Calvert, Rhonda Ryman, and Ilene Fox. [n. d.]. From dance notation to human animation: The LabanDancer project. *Computer Animation and Virtual Worlds* 16, 3âĂŘ4 ([n. d.]), 201–211. https://doi.org/10.1002/cav.90 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cav.90

[14] Xueyan Zhang, Zhenjiang Miao, and Qiang Zhang. 2018. Automatic Generation of Labanotation Based On Extreme Learning Machine with Skeleton Topology Feature. In *2018 14th IEEE International Conference on Signal Processing (ICSP).* IEEE, 510–515.

A DESCRIPTION FILES

The input into the Salsational graphical system is a set of two data exchange files, a Move file and Sequence file (specified in YAML format), and motion data files. A Move file contains a reference for the moves of a variation of a dance style. Each reference consists of the name of the move, an optional description of the move, a reference to the file path of the motion file where the movement occurs, and either a pair of integers specifying the beginning and ending frames that show the move, or raw transformation data between the two frames that can be used directly to render the move. Move files can act as local or global references, as specified by the user. A Sequence file specifies valid sequences of moves, where each move is derived from the associated Move file. The use of global Move files facilitates a standardized knowledge-base that every user shares and can learn from. It also allows movements for a dance style to be specified once, and unique sequences to be derived from the file.

A.1 Move File Specification

- dance_style: defines the style of dance
- variation: the variation of the style of dance
- source_method: either external or inline
- moves: an array of move objects
  - ref: a unique identifier for this move
    - name: the name of the move
    - desc: an optional description of the move
    - src_ref: the relative file path to the motion data file
    - begin_frame: the beginning frame of the move
    - end_frame: the ending frame of the move

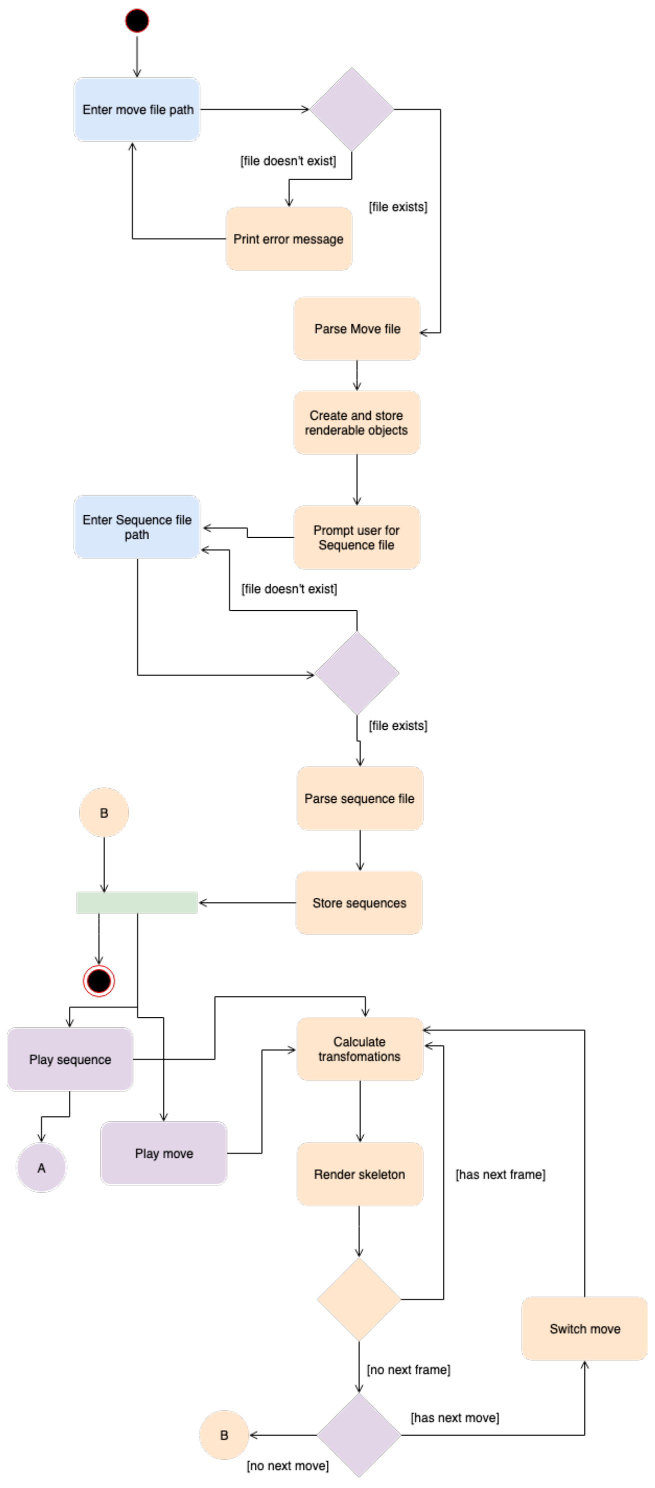

Figure 17: Activity diagram showing the high-level functioning of the Salsational system

A.2 Sequence File Specification

- name: the name of the sequence file
- mov: the file path to an associated Move file
- sequence
  - ref: a unique identifier for the sequence
    - name: the name of the sequence
    - moves: the list of validated moves in the sequence

When a Sequence file is loaded, the system locates the associated Move file and retrieves the list of moves defined in each sequence.

The system then parses the motion data for each move and calculates the transforms for each object, stopping at the ending frame. Each transform is then stored locally and can be referenced by other components until run-time completes in order to prevent redundant reloading of the same file.

B SUPPLEMENTARY INFORMATION

B.1 Process View Activity Diagram