

FeynGame

R.V. Harlander, S.Y. Klein, M. Lipp

Institute for Theoretical Particle Physics and Cosmology, RWTH Aachen University, 52074 Aachen, Germany

Abstract

A java-based graphical tool for drawing Feynman diagrams is presented. It differs from similar existing tools in various respects. For example, it is based on models, consisting of particles (lines) and (optionally) vertices, each of which can be given their individual properties (line style, color, arrows, label, etc.). The diagrams can be exported in any standard image format, or as PDF. Aside from its plain graphical aspect, the goal of FeynGame is also educative, as it can check a Feynman diagrams validity. This provides the basis to play games with diagrams, for example. Here we describe on such game where a given set of initial and final states must be connected through a Feynman diagram within a given interaction model.

PROGRAM SUMMARY

Program title: FeynGame

Distribution format: gzipped tar archive, GitLab repository Authors: R.V. Harlander, S.Y. Klein, M. Lipp Licensing provisions: GNU General Public License 3 (GPL) Programming language: Java Operating systems: Linux, Windows, MacOS Keywords: Feynman diagrams, Java, GUI Nature of problem: Efficient drawing of Feynman diagrams for presentations and publications; playful elementary introduction to the concept of Feynman diagrams Method of solution: Graphical interface which incorporates the Feynman rules for various models of particle interactions. Restrictions: Only the topological information of the Feynman rules is incorporated. Running time: The running time depends on the diagram to be drawn and the skill and practice of the user.

1 Introduction

Feynman diagrams are a central tool for particle physics [1]. Not only are they indispensable for perturbative calculations of observables such as cross sections or decay rates. They are also extremely useful in every-day scientific communication, from the professional research to the educational and even the popular science level (see, e.g., Ref. [2, 3]). In this way, they provide an excellent bridge to convey scientific knowledge to a broad audience.

A perturbative calculation for a given process within a specific model (e.g. the Standard Model (SM)) typically starts with the generation of the relevant Feynman diagrams. This is an algorithmic process which can be implemented into a computer program. The most prominent examples for such implementations are FeynArts [4, 5] and qgraf [6]. The diagrams are then usually passed on to other programs which translate them into mathematical expressions according to the so-called Feynman rules, and yet again to other tools which actually evaluate these expressions. A paradigm example where all these steps are incorporated up to next-to-leading order (NLO) in perturbation theory in a single framework is MadGraph [7].

In all of this process chain, the actual visualization of the diagrams is not required, since in the ideal case no human intervention is necessary. Only the computer "sees" the diagrams, albeit in some mathematical encoding such as incidence matrices. Nevertheless, communication about particle physics still relies heavily on Feynman diagrams, for example in order to refer to a particular process or specific contributions to it (see, e.g., Ref. [8]). On the other hand, their one-to-one correspondence to Feynman *integrals* also increases the readability of relations among these integrals. For example, a famous identity among massless scalar integrals takes the form [9]

$$\epsilon \xrightarrow{\mathbf{q}}_{\mathbf{p}} = \underbrace{\mathbf{q}}_{\mathbf{p}} \xrightarrow{\mathbf{q}}_{\mathbf{k}} - \underbrace{\mathbf{q}}_{\mathbf{p}} \xrightarrow{\mathbf{q}}_{\mathbf{k}}, \qquad (1)$$

which the expert reader immediately translates into Feynman integrals:

$$\epsilon \int \frac{\mathrm{d}^{D} p \,\mathrm{d}^{D} k}{p^{2} k^{2} (k-q)^{2} (p-q)^{2} (k-p)^{2}} = \int \frac{\mathrm{d}^{D} p \,\mathrm{d}^{D} k}{p^{2} k^{2} (k-q)^{4} (k-p)^{2}} - \int \frac{\mathrm{d}^{D} p \,\mathrm{d}^{D} k}{p^{2} k^{2} (k-q)^{2} (p-q)^{4}}$$
(2)

which holds for space-time dimensions $D = 4 - 2\epsilon$, with $\epsilon \neq 0$.

For these reasons, computer tools for the actual *visualization* of Feynman diagrams are important. It is probably fair to say that the discussion about the most suitable tool for this purpose is one of physicists' favorite (non-scientific) controversial subjects. It seems that the optimal compromise between handiness, flexibility, and esthetics has not been achieved yet by any of the existing computer tools that have been designed for this purpose.

The latter naturally fall into two categories: text-based and graphical tools. The former usually provide a set of routines and statements ("packages") within a framework like LATEX or C++ which encode graphical objects. The user needs to write some code which, upon compilation, produces a visual image of the diagram. Graphical tools, on the other hand, provide a graphical user interface (GUI) which allows the user to directly draw the diagrams onto a "canvas" with the "mouse".¹

The earliest examples for text-based tools are probably FeynDiagram [10], which is a set of C++ objects for producing Feynman diagrams in PostScript format, or axodraw [11] and FeynMF/FeynMP [12, 13], both of which allow to include the code directly into the source of a LATEX document. The diagram is then generated by compiling the LATEX code, possibly with accompanying runs of additional programs such as metafont, metapost, or ps2pdf. More recent tools are axodraw2 [14] and PyFeyn [15] while currently the most popular text-based system seems to be TikZ-Feynman [16], which uses lualatex [17] to automatically position the vertices properly.

The most widely used tool which includes a GUI is probably JaxoDraw[18], which is a java interface for axodraw. Recently, a few other graphical tools have appeared, most notably feynman[19] and Feynman diagram maker [20], both of which can be run online from within a browser.

All of the tools above, whether text- or GUI-based, are designed to draw individual Feynman diagrams "line-by-line". There are only very few tools that automatically visualize the output of Feynman diagram generators. The prototype of such tools has been FeynArts [5, 4], which can even be combined with the FeynEdit GUI [21] in order to feyn-, sorry: fine-tune the output of FeynArts (it can also be used stand-alone).

The only tool to produce animated Feynman diagrams that we are aware of is aximate [22].

Rather than doing a detailed comparison of FeynGame with existing tools, let us simply state what we believe are its most important, and in particular its unique features. The purpose of FeynGame is two-fold: on the one hand, it should allow to draw Feynman diagrams of high quality in a very fast manner. It is thus well-suited for quickly supplying Keynote or PowerPoint presentations with Feynman diagrams through a simple cut-and-paste operation. By exporting the diagrams as PDF, they can of course be included in type-set documents such as books or scientific papers as well.

¹We use the expressions "mouse", "mouse wheel", "click" etc. in a generic way, referring to all kinds of "mouse-like" input devices such as track pads or tablet pens, and the corresponding control actions.

The second purpose of FeynGame is educational: since its usage is based on particle models (QED, SM, etc.), it can check whether a particular Feynman diagram is consistent with the underlying model. This provides the basis for playing games with Feynman diagrams. The current version of FeynGame contains the game InFin, where the player is asked to turn a random pair of initial and final states into a valid connected Feynman diagram. We believe that this may help to bring the concept of FeynGame will include further games.

FeynGame is used through a GUI written in java. It is therefore highly portable and should run on any platform which provides java (version 8 or later). Even though FeynGame should work out-of-the-box, it is designed to be personalized. This means that the user can provide her own *model file* which defines all the desired (or required) line and vertex styles. This avoids the need to modify every individual line or vertex in case its appearance is required to deviate from the overall default. For example, FeynGame provides a *model file* for the SM which contains a unique line for each particle of that model. To draw a particular Feynman diagram, one simply picks the individual lines which are displayed in the main window of FeynGame. The user can conveniently change the overall appearance of these lines, either through the GUI itself, or by editing the *model file*.

Another unique feature of FeynGame is the automatic splitting of a line once another line is connected with it through a vertex. This greatly facilitates the iterative drawing of diagrams, i.e. adding lines to lower order diagrams to create higher order effects through internal loops or real radiation. Lines can be moved, stretched, or rotated using a single click-and-drag, the curving of lines is achieved by turning the mouse wheel, their direction can be changed and labels can be added with a single keystroke. Almost all line parameters can be adjusted either through keyboard shortcuts, via a menu (the *EditFrame*), or by editing the *model file*.

The state of the canvas (and thus the current diagram) can be stored in an internal format (.fg files) which includes up to the last thousand steps in the history of the current state. This allows to undo/redo any modification of the diagram, no matter when the diagram was originally created. The current diagram is stored automatically, so that closing and re-opening FeynGame allows one to seamlessly continue working on the current session.

A diagram can be exported as an image in various formats to a file, it can be copied to the clipboard (which allows to simply paste it into presentation tools like PowerPoint or Keynote), and it can be turned into vector graphics in PDF format, which is useful for including it in a typeset document as produced by LATEX or Word.

The current document gives a first introduction to FeynGame, roughly following a stepby-step procedure. For simplicity, we will assume that FeynGame was downloaded from



Figure 1: Drawing a diagram for $b \to s\gamma$. Step (i): draw a *b*-line. Step (ii): draw a *W* line. This splits the original *b*-line into three. Step (iii): curve *W* boson line (mouse wheel). Step (iv): add a photon line and curve it slightly. Step (v): using *EditFrame*, turn the central *b*-line to a *t*-line. Step (vi): using *EditFrame*, turn the left *b*-line to a *s* line (The label positions have been slightly adjusted at intermediate steps.)



Figure 2: The default main window of FeynGame, showing the menu bar with items File, Edit, View, and About, the canvas below it, and the default model bar at the bottom with tiles for the t, g, γ , and H line, and the "+" tile to add new objects to the model.

this URL^2 in the form of a tarball. Unpacking the tarball will produce a folder named feyngame, with the following structure:

1	img/	java/	levels/	models/
---	------	-------	---------	---------

We assume that the user's current directory is feyngame.

2 Draw Mode

Let us start FeynGame without command line arguments and without loading a *model file* (see Sect. 2.4). The call from a terminal will thus look something like

1 \$ java -jar java/FeynGame.jar

while on many systems (e.g. Windows, Linux) double clicking FeynGame.jar will work as well. Either way, this will open a dialogue window which allows one to choose between "drawing mode" and "game mode". Let us consider drawing mode first.

Table 1: The three main auxiliary features of FeynGame. They can be toggled by pressing the corresponding key, or via the menu item View, which also allows one to adjust the grid size.

feature	key	description
grid	g	equidistant grid of points on the canvas
helper lines	h	indicate the active region of an object
show active object	а	moving dash pattern for active object on the canvas

2.1 The main window

Selecting the drawing mode, FeynGame will open only the "main window", shown in Fig. 2. It contains the "canvas" on which the Feynman diagram will be drawn, and below it a number of tiles. The latter define graphical objects (the *current model*) which can be used to draw diagrams, except for the left-most tile (showing a big "+") which allows to add new objects to the *current model* (see Sect. 2.4). The elements of the menu bar at the top of the main window will be described in the course of this paper, whenever the corresponding functionality is discussed.

By default (i.e. without loading a *model file*, see Sect. 2.4), FeynGame will use a toy model as the *current model*, consisting of only a few particles which we will refer to as quark, gluon, photon, and Higgs, respectively (see the tiles of the main window in Fig. 2). It is one of the central ideas of FeynGame that this default model can be replaced by a personalized model via File Load model file in the main menu, see Sect. 2.4 below. This *model file* will then define the *current model*.

For the moment, we will assume that the three main *auxiliarly features* of FeynGame, described in Table 1 are de-activated. Thus, if your canvas shows a grid of points, press g to switch off the *grid* feature for the moment.

2.2 Single line

Let us select a quark line by a single click on the corresponding tile. Subsequent clickand-drag on the canvas will draw a straight quark line between the click and the release point, see Fig. 3 (a). If that line shows a moving dash pattern, press **a** to switch off the *show active object* feature for the moment. If you hover the mouse pointer over that line (without clicking), the width of that line increases slightly; this indicates that clicking at this point will grab the line. If part of the line changes color when hovering over it, press

²http://www.robert-harlander.de/software/feyngame



Figure 3: (A) the canvas contains a single t line; (b) adding a γ line close to it results in two t lines, a γ line, and a vertex marker; (c) moving the γ line away from the vertex results in a single t line and a single γ line; (d) moving the vertex in (b) moves also the ends of the lines connected with it.

h to switch off the *helper lines* feature for the moment.

You can now modify this line in several ways. Let us first discuss geometrical modifications which are most conveniently applied by operating directly with the mouse on the canvas:

- move: You can either move the entire line by grabbing the line close to its center, or only one end of the line by grabbing it in the vicinity of this end. Hovering over the line, FeynGame will display a small textbox in the upper left corner of the canvas which tells you whether grabbing the line at this point will move the entire line or just one end of it. Switching on the helper lines feature (toggle with h, see Table 1) will indicate this directly by highlighting the "active region" of the line when hovering over it with the mouse pointer.
- *curve*: Turning the mouse wheel will curve the line. Of course, after curving, you can again move the line in the same way as above. Moving an end of a curved line will preserve the height of the corresponding circular segment, which means that the radius of the segment will change. This makes it easy to draw closed circles, see Sect. 2.8.6.
- *invert*: You can invert the "direction" of this line by pressing i. For the quark line, this will invert the direction of the arrow on the line. For photons and gluons, it will invert the wiggles/spirals. The position of the line label will change accordingly (see Sect. 2.7).

Other line properties like color, stroke size, or arrow size may preferably be changed using the EditFrame. This is a separate window which can be opened (and closed) by selecting an object on the canvas and pressing $[\bullet]$, or via the View menu item. The actual content

Gluon		Color		
Inve	rt line	Double line		
Stroke size	2 🛊	Curvature	0	 ▼
Wiggle height	20 🗘	Wiggle length	27	* *
Wiggle offset	15 🚔	Dashed	1	A V
X (foot)	181 🛊	X (head)	422	•
Y (foot)	197 🛊	Y (head)	36	*
🗹 Show label: g				
Rotation:	0 🍃 So	ale:	2	*
Part of path	0,515 🖨 D	istance to line	28	*
	Set as default l	abel parameters		
t	g	∼ v	н	

Figure 4: The *EditFrame* window for a gluon line.

of the *EditFrame* window depends on the object that shall be modified; Fig. 4 shows the *EditFrame* window for gluon lines, for example. It also allows to change the relevant parameters which determine the form of the wiggles. Furthermore, one may attach a text label which will retain its relative position to the line even when it is moved or curved, see Sect. 2.7.

For most parameters and options accessible through EditFrame there are keyboard shortcuts. A complete list can be displayed through the menu item View Show keyboard shortcuts. On the other hand, the position, curvature and orientation of a line can also be controlled through EditFrame, which even overrides the presence of the grid.

Any object on the canvas can be removed by selecting it and pressing backspace, or via the menu item Edit Delete active object.

2.3 Connecting lines

Let us now pick a photon line by selecting the corresponding tile below the canvas. Again, click-and-drag will add that line to the canvas. If one of the end points of the new line is close to the already existing quark line, the photon line will be connected with it, indicated by a vertex marker (the black dot), see Fig. 3 (b). In this process, the quark line is split into two adjacent quark lines, each inheriting the properties of the original quark line (except for the length, of course). The canvas now contains four objects: two quark lines, one photon line, and a vertex marker. Clicking on one of these objects will make it "active"; the *EditFrame* window will always refer to the active object. Switching on the *show active*



object feature (toggle with a), see Table 1) will indicate the active object through a moving dash pattern.

Each of the three lines can be modified in the same way as above. Removing the photon line from the canvas (see Sect. 2.2), or detaching it from the vertex will cause the two quark lines to re-combine to a single quark line if they still have the same essential properties such as color, width, angle, etc.³ and the vertex marker will disappear, see Fig. 3 (c).

Moving the vertex, on the other hand, will move all ends of the lines connected with it, see Fig. 3 (d). This is a convenient feature for fine tuning of Feynman diagrams. Other properties of the vertex that can be modified are accessible by activating it (i.e. clicking on it) and opening *EditFrame* (i.e. pressing e).

2.4 Model file

As described above, *EditFrame* allows one to modify the appearance of lines and vertices. However, it is more along the philosophy of FeynGame to use a pre-defined model (say, QED, the SM, the Minimal Supersymmetric SM (MSSM), or a subset of the associated particles) where each particle corresponds to a line with a certain appearance. For example, if we are only interested in weak interactions of the first two generations of leptons, the main window of FeynGame could look like Fig. 5. The leading-order diagram for muon decay can then be simply drawn by selecting a muon line, drawing it on the canvas, followed by the line for a W boson, a muon neutrino ν_{μ} , an electron, and an electron neutrino ν_{e} . It should not be necessary to change the line styles or manually add particle labels.

Drawing a printable version of a diagram thus literally becomes a matter of seconds, because the different line styles for the individual particles have been defined beforehand, using the *model file*. The *model file* for the example above reads

³Text labels will be combined if they differ.

Table 3: Line options.				
option	value	applies to		
color	color name, hex	all		
label	string, html	all		
stroke	pixels	all		
dash	$\mathrm{true}/\mathrm{false}$	all		
dashLength	pixels	all		
arrowSize	pixels	fermion		
wiggleSize	pixels	photon, gluon		
wiggleHeight	pixels	photon, gluon		
wiggleOffset	pixels	gluon		
wiggleSharpness	pixels	photon		
double	$\mathrm{true}/\mathrm{false}$	all		

1 [e, E, fermion, color=Red, label=e]
2 [nue, Nue, fermion, color=dc27e7, label=<html>&nu<sub>e]
3 [mu, Mu, fermion, color=Green, label=<html>&mu]
4 [numu, Numu, fermion, color=27e7b3,label=<html>&nu<sub>&mu]
5 [W, W, photon, color=dc27e7, label=W]
6 [Z, Z, photon, color=Blue, label=Z]

```
7 [ ph, ph, photon, color=Green, label=<html>&gamma ]
```

For future reference, let us assume that these lines are the content of a file named EW2gen.model. Each pair of square brackets defines a particle. The first two entries assign an internal name for the particle. Since fermion lines have a direction, the two entries are different in this case (e.g. e, E for the electron, or nue,Nue for the electron-neutrino), while they are identical for bosons (W,W or Z,Z). The third entry defines the basic line style according to Table 2. These three entries are required when defining a line in the *model file*. The label parameter is optional and will be discussed in more detail in Sect. 2.7. Other optional parameters are listed in Table 3.

Starting FeynGame by giving it the name of the *model file* as argument (including its path) will load the specific model instead of the default one. For example, if we say

1 \$ java -jar java/FeynGame.jar models/EW2gen.model

FeynGame will directly start in drawing mode, using the model defined in EW2gen.model as the *current model*, with the main window as shown in Fig. 5. Alternatively, one can change the *current model* from the main menu of FeynGame using File Load model file.

Aside from the default, built-in model, FeynGame comes with model files for QED, the SM,



Figure 5: FeynGame's main window when called with EW2gen.model.

and also with EW2gen.model, see Fig. 5. The user may adjust the parameters of these models to ones personal taste simply by editing the corresponding model file. Another, sometimes more convenient option is to modify the appearance of an existing line within FeynGame (for example by using *EditFrame*), and then selecting the "+" tile in the main window. This will add a new object tile to the *current model* at the bottom of the main window. The new line is thus available as a new object in FeynGame. Selecting File Save model file (as) will modify the current *model file* (or create a new one), so that the new line is available also in future sessions. Similarly, one may remove existing object tiles by $\widehat{\ }$ -clicking the tile and subsequently saving the *model file*.

A third way to modify the *model file*, which is a mixture of the two options just discussed, will be described in Sect. 2.8.3.

2.5 Vertices

Optionally, you may also specify the vertices of a model in the model file. Unless you assign specific markers to these vertices (see Sect. 2.6 below), this has no effect on the actual drawing or appearance of a diagram. However, if the vertices of the model are specified, you can ask FeynGame to check whether the specific diagram on the canvas is consistent with the current model. For example, assume that you add the following lines to EW2gen.model:

1 {e, E, ph}
2 {mu, Mu, ph}
3 {e, E, Z}



Figure 6: (a) A "wrong" diagram for μ decay. (b) Pressing f reports the errors.

4 {mu, Mu, Z}
5 {nue, Nue, Z}
6 {mue, Mue, Z}
7 {e, Nue, W}
8 {nue, E, W}
9 {mu, Numu, W}
10 {numu, Mu, W}
11 {W, W, ph}
12 {W, W, Z}

This tells FeynGame the topological Feynman rules for the electro-weak interaction of the first two lepton generations. It will still allow you to draw any Feynman diagram you like, for example the one shown in Fig. 6 (a). However, by pressing f, FeynGame will report that this diagram is not consistent with the current model by displaying the message in Fig. 6 (b).

The main purpose of defining the vertices of a model is in the context of the game mode of **FeynGame** though, see Sect. 3. However, aside from the simple consistency check described here, it may also be useful to assign specific vertex markers to some of the vertices. This is described in the next section.

2.6 Vertex markers

As described in Sect. 2.3, FeynGame draws a vertex marker at the point where lines are connected with one another. Using *EditFrame*, the style of this marker can be modified.



Figure 7: Triple gauge boson vertices are marked in red.

By selecting the "+" tile, an object tile for the currently active vertex marker will be created. Similar to newly created lines, the new marker style can even be saved to the *model file* using File Save model file, so that it is available as a *free-floating object* for future sessions when calling FeynGame with this file (see below). The *model file* format for a marker is

```
1 (vname=redmark, size=10, borderColor=ff000000, borderDashed=false,
borderDashLength=5, borderStroke=1, fillingColor=Red)
```

This would define a big red vertex marker in FeynGame. The meaning of the parameters should be self-explanatory. If the parameter vname is missing or is equal to the string "default", FeynGame will use this marker as default.⁴ Otherwise, it will use the internal default marker. On the other hand, you can assign a specific marker to an individual vertex by adding the attribute type to the vertex definition. For example, modifying the triple gauge boson vertices defined above in EW2gen.model according to

```
1 {W, W, ph, type=redmark}
2 {W, W, Z, type=redmark}
```

will use the red marker for these vertices, and the default marker for all others, see Fig. 7.

The new marker will now also appear as a tile object in the FeynGame main window. It can be placed anywhere on the canvas as a *free-floating object*, in the sense that it will not

⁴If there are multiple vertices without the **vname** parameter or it being equal to "default", the uppermost of these vertex definitions is used as the default vertex.

Table 4: Ja	<u>waFX ve</u>	ertex markers.
f	xpath	vertex marker
cro	ss.fx	\otimes
hatch	ed.fx	
crosshatch	ed.fx	igodol
sta	ar.fx	\bigotimes

be connected with any other object (it does "clip" to grid points though, see Sect. 2.8.1).

FeynGame provides a set of more elaborate vertex markers. They are accessible through the drop-down menu Select filling from default patterns of the vertex marker *EditFrame*, or by adding the fxpath option to the vertex definition in the *model file*. For example,

1 (fxpath = cross.fx)

will provide a vertex marker with a cross in the middle. A list of available vertex markers and their corresponding fxpath parameter are shown in Table 4.

2.7 Labels

As indicated in Sect. 2.4, it is possible to attach labels to lines. In the *model file*, this is done by including the label=<label> keyword in the line definition, see EW2gen.model. Currently, labels can be given in plain text format, or using basic HTML commands. In this case, the label definition should start with the string "<html>" (see the definition of the ν_e label in EW2gen.model, for example). Using the <html> combined with the tag, it is possible to change the color and the font of the text.

The position of a label is relative to its associated line, but it can be changed by mouse or using EditFrame. Moving the line will also move the label, such that the relative position of the two remains fixed. On the other hand, the orientation of the label is defined relative to the canvas, i.e. rotating the line will not rotate the label relative to the canvas. The *default* properties of the label (position, size, orientation, etc.) are the same for all lines; they can be changed by pressing the **Set as default** button in *EditFrame*. In this way, one can also determine whether the label is shown or not by default. Pressing I will toggle the displaying of the active line's label.

In the same way, labels can also be attached to vertex markers and other objects.

2.8 Other features

2.8.1 Grid

If the grid is switched on (toggle with g, see Table 1), all objects except for line labels will be "clipping" to this grid.⁵ This means that the endpoints of lines, or the center of *free-floating objects* (see Sect. 2.6), can only be placed at the grid points. The grid points are shown on the canvas, but are not included when the diagram is exported as an image or a PDF. This also holds for the other auxiliary options like *helper lines* or *show active object*. The spacing of the grid points can be changed from the menu $View \ln(De)$ crease grid spacing, or through the associated keyboard shortcut, see View Show keyboard shortcuts.

In addition to the visible grid points on the canvas, every line on the canvas introduces a "local" set of grid points along this line, roughly at the same distance as the visible grid points on the canvas. If the *grid* is active, lines can be connected with one another only at these line-specific grid points.

Sometimes one may want to have an object to be located off the grid. This can be achieved either by temporarily switching off the grid (e.g. by pressing g). An object placed without the grid will remain at its (possibly off-grid) position even after the grid is switched back on. The other option to place an object off the grid is via *EditFrame*, which allows one to specify the coordinates of the object in pixels.

2.8.2 Clipping

Independent of whether the grid is on or off, FeynGame will do additional kinds of clipping when objects are moved by mouse or curved by mouse wheel:

- **Vertex clipping.** If the end of a line is moved closer to another line (or to its other end) than a certain minimal distance, it will be connected with that line through a vertex.
- **Curvature clipping.** If the curvature of a line is decreased below a certain minimal value via the mouse wheel, it will be set to zero.
- Angle clipping. If the slope of a straight line is close to a multiple of $\pi/2$, it will be replaced by that multiple of $\pi/2$. Via the menu item Edit Clipping angles or by repeatedly pressing c, that value can be changed to $\pi/4$ or $\pi/16$, or zero.

This kind of clipping is introduced for the user's convenience, because otherwise it would be rather clumsy to connect lines, or to turn a curved line into a completely straight line.

 $^{^{5}}$ This applies only to newly positioned objects. The position of object already on the canvas will not be altered by switching on the grid.

In order to overrule such clipping, one can use *EditFrame* to do a pixel-wise modification of the position or the curvature.

2.8.3 Copy/Paste

Objects can be copied and pasted using Edit Copy and Edit Paste from the main menu, or through the system-defined keyboard shortcut ($\frac{2}{2}-\frac{2}{2}$ for MacOS, for example). The object will be pasted to the center of the canvas (if pasted from menu) or close to the mouse pointer (if pasted from keyboard).

One may also copy an object from the canvas to the clipboard, and then paste it to a regular text editor. It will then display the model-file format of that object. Vice versa, one may paste any line from the model file into the FeynGame canvas, and it will show the graphical representation of that object.

2.8.4 Text

There is no specific text object in FeynGame. As described above, labels of lines and vertices are attributes of these objects. If one really needs a separate text element which is not associated with any visible object, one may define a vertex of size zero, and introduce the text as the vertex label, e.g.

(vname=text, size = 0, label=mytext)

2.8.5 Images

There are different ways to include an image on the canvas. The first one is to define it as a vertex marker, for example by opening *EditFrame* for an arbitrary vertex on the canvas, and clicking Select image / JavaFX file, which opens the file chooser, from where one may select an arbitrary image (JPG, PNG, GIF or BMP). One can then include it in the *current model* with the "+" tile. Or one defines it directly in the model file; the syntax is:

```
1 (imagePath=<path_to_image>/Unicorn.jpg)
```

Other options like size, rotation, etc. are available but not shown here. Note that this method puts the image into a round shape, which means that part of it may be cut away. However, one can use this object like an other vertex marker; for example, one may associate it with a specific vertex as described in Sect. 2.6.

Another way to include an image is to simply paste it from the clipboard into the FeynGame canvas, using the system-defined keyboard shortcut. For example, in MacOS one could use



Figure 8: A circle can be drawn by identifying the ends of a curved line (a)–(c). Attaching a line to the circle turns it into a tadpole (d); the vertex becomes the new "glue point".

 \mathbb{H} -ctrl- \mathbb{D} -4 to take a screen shot and paste it into the FeynGame canvas using \mathbb{H} -v. Again, the image can be added as an object to the *current model* using the "+" tile. When stored to the model file, it will appear there as a line of the following form:

1 |imagePath=<path>/<image>.png, rotation=0, scale=1|

The options should be self-explanatory.

2.8.6 Circles and Tadpoles

We define a circle diagram as a closed loop which is not connected with any other line. It can be obtained by connecting the ends of a curved line with each other, see Fig. 8 (a)–(c). One can "re-open" the circle and turn it back to a regular curved line simply by click-and-drag in the vicinity of the "glue point". Once we connect a (single) line with the circle, it becomes a tadpole diagram, see Fig. 8 (d). While the actual position of the original circle does not change, its glue point gets identified with the new vertex.

2.9 Saving and Exporting

Saving a diagram. FeynGame automatically saves the status of the canvas across sessions in an internal format (.fg file), including its history.⁶ This means that if you open FeynGame, the session will start at the point where it was previously closed. Selecting $\boxed{\mathsf{File}}$ New from the menu, on the other hand, will clear the canvas and the history.

Selecting File Save (as) from the menu allows you to save the status of the canvas, including its history, in the internal format to a specific file. You can modify the diagram at a later stage by opening it in the canvas using File Open. Selecting Edit Delete history will delete

 $^{^{6}\}mathrm{Up}$ to the last thousand steps.

the history of the canvas. Storing a diagram without history typically reduces the size of the .fg-file significantly.

Export as image. A diagram can be exported to an image file in various formats using File Export as image from the menu. None of the auxiliary features from Table 1 will be exported, only the plain diagram will be visible. Exporting as "Portable Network Graphics (*.png)", the background of the diagram will be transparent. In addition, the diagram can be exported to the clipboard using File Export to clipboard. In this case, the background of the image will be transparent. This is particularly useful for including the diagram into a PowerPoint or Keynote presentation.

Exporting as PDF/printing. With File Print / to PDF, a diagram can be exported as PDF, or directly sent to a printer. FeynGame will open a dialog box which controls the following features, see Fig. 9 (b):

- *automatic boundaries:* The bounding box of the PDF is determined automatically from the actual location of the diagram on the canvas, roughly as indicated by the dotted black frame around the diagram in Fig. 9 (a).
- print full canvas: The bounding box is determined by the size of the canvas.
- Show parameters of bounding box: The bounding box parameters are displayed when the diagram is printed or exported, see Fig. 9 (c). They can then be copied to the clipboard and pasted into a text editor, for example as parameters for the \includegraphics command in IAT_EX :

```
1 \includegraphics[viewport = 18 347 577 824,
2 size=.3\textwidth]{ggh.pdf}
```

Equivalently, one can use the trim parameters, see Fig. 9 (b).

• Print in B/W: Print in black-and-white. More precisely: Lines and the borders of vertices are drawn in black. The interior of the vertices is drawn black or white depending on the brightness of its actual color. All other objects remain untouched.

2.10 Misc

Object layering: Lines on the canvas are layered according to their activation history, i.e. the line that was active last is at the top, etc. This also holds for their labels. Vertex markers, including their labels, are always on top of everything.



Figure 9: Printing to PDF. (a) Diagram to be printed using File Print / to PDF; the bounding box is indicated as a black dotted frame in this picture. (b) Dialogue box with bounding box and black-and-white options. (c) Bounding box information.



Figure 10: FeynGame in game mode. (a) the "challenge" is to connect the initial (left) with the final state (right) through a connected Feynman diagram which only uses lines and vertices of the underlying model. (b) a possible solution.

- **Canvas shifting: ①**-Click-and-drag on empty space on the canvas will move the whole diagram.
- **Right-clicking:** Right-clicking anywhere on the canvas opens a menu next to the mouse pointer which provides an alternative to selecting objects by clicking on the tiles in the main window. Right-clicking close to an object in addition provides the possibility to change the style of this object, as an alternative to using the *EditFrame*.

3 Game mode

3.1 Idea of the game

Currently, FeynGame provides a single game mode, called InFin (for <u>Initial-Fin</u>al). It can be started by passing a *level file* to FeynGame upon start-up:

```
i java -jar java/FeynGame.jar levels/SMLevel.if
```

Alternatively, one can start FeynGame without any argument, choose InFin from the dialogue window, and subsequently specify the *level file*. With the *level file* SMLevel.if provided with the FeynGame distribution, the main window typically looks similar to Fig. 10. The lines on the canvas indicate the initial and the final state of a process. The goal is to draw a connected Feynman diagram that contributes to this process, using the Feynman rules of the current model (the SM in this case). An example for a valid diagram is shown in Fig. 10 (b). Pressing Finish will add the "Points of this challenge" as indicated below the canvas to the total number of points. If the diagram is not valid, pressing Finish will report the errors. Choosing "Use a timer with duration in seconds" in the dialogue box upon starting FeynGame in game mode will restrict the total available time for the game to the specified number of seconds. This can also be achieved by passing the number of seconds as an integer to FeynGame when calling it from the command line like above.

When drawing the diagram, it is important to note that the outer end of each external particle is tied to the border of the window and cannot be moved; the other end has to be connected with some other particle (external or internal). Also, this is *the only way* to connect an external particle with the diagram, i.e., other particles cannot be connected with external lines anywhere else but at this "inner end".

3.2 Level file

A sample *level file* looks like this:

```
1 model = ../models/standard.model
2
3 easy
4 start: e,E
5 end: e,E
6 start: e,e
  end: e,e
7
8
9 medium
10 start: H
11 end: Z,g,g
12 start: mu
13 end: e,numu,Nue
14
15 hard
16 start: b
17 end: s,ph
18 start: b,S
19 end: B,s
```

Of course, FeynGame needs to know the underlying model. For that purpose, one needs

to point to a proper *model file*, including the vertex information, whose structure is as discussed in Sect. 2.4 and 2.5. This is done in line 1; the path to the file can be relative to the path of the *level file* or absolute.⁷ The rest of the *level file* is divided into three blocks, headed by the keywords "easy", "medium", and "hard". Below each of these headings is a list of processes, specified by their initial state and final state (start and end, respectively). Easy/medium/hard processes are assigned 5/15/25 points.

The user may modify the *level file* by adding or removing processes, changing the model file, etc. In principle, given a *model file*, the generation of the *level file* could be automated, of course; this is currently work in progress.

3.2.1 Directories and Files

As indicated above, FeynGame automatically stores the current state of the canvas anytime during the session. It will report the location of the corresponding files upon exit. For example, in MacOS, it prints the following message on the terminal:

```
    Saved preferences to: <home>/Library/Preferences/FeynGame/DrawMode.ini
    Saved diagram to: <home>/Library/Preferences/FeynGame/last.fg
```

While last.fg is a binary file which contains the current status of the canvas, including its history, DrawMode.ini is a regular ASCII file which lists the current parameters of the session. They will be applied upon the next start of FeynGame. While it is not recommended, one may thus modify the parameters in DrawMode.ini using a text editor. However, it is usually simpler (and safer) to adjust the global parameters from within FeynGame.

4 Conclusions and Outlook

The purpose of FeynGame is two-fold: On the one hand, it should be a useful tool for the efficient and intuitive drawing of Feynman diagrams for presentations and publications. On the other hand, it serves a didactic purpose, in the sense that it should playfully convey the concept of Feynman diagrams to non-experts. The basis for this is its ability to check the validity of a certain Feynman diagram w.r.t. an underlying model. The current version of FeynGame contains one such game, called InFin, which is close to one of the main actual applications of Feynman diagrams, namely the theoretical description of a scattering or decay process. An example for another possible game is a Scrabble-type game where valid Feynman diagrams must be created from randomly emerging lines.

⁷As usual, the home directory can be abbreviated by the character "~" on Linux-like systems.

Other future plans for FeynGame include the im- and exporting of Feynman diagrams to text-based programs like TikZ-Feynman [16], and the possibility to include LATEX text.

FeynGame is published as open source under the GNU General Public License. It can be downloaded from https://gitlab.com/feyngame/FeynGame, or as a gzipped tar ball from https://web.physik.rwth-aachen.de/~harlander/software/feyngame.

Acknowledgments. We would like to thank Jonas Klappert, Fabian Lange, Magnus Schaaf, Nils Schöneberg, and Małgorzata Worek for helpful comments.

References

- [1] R.P. Feynman, The Theory of positrons, Phys. Rev. 76 (1949) 749.
- [2] D. Kaiser, Drawing theories apart: The dispersion of Feynman diagrams in postwar physics, University of Chicago Press, 1993.
- [3] A. Wüthrich, The Genesis of Feynman Diagrams, Springer, 2010.
- [4] T. Hahn, Generating Feynman diagrams and amplitudes with FeynArts 3, Comput. Phys. Commun. 140 (2001) 418, hep-ph/0012260.
- [5] J. Küblbeck, H. Eck, and R. Mertig, Computeralgebraic generation and calculation of Feynman graphs using FeynArts and FeynCalc, Nucl. Phys. Proc. Suppl. 29A (1992) 204.
- [6] P. Nogueira, Abusing qgraf, Nucl. Instrum. Meth. A559 (2006) 220.
- [7] J. Alwall, M. Herquet, F. Maltoni, O. Mattelaer, and T. Stelzer, MadGraph 5: Going Beyond, JHEP 06 (2011) 128, arXiv:1106.0522 [hep-ph].
- [8] M. Stöltzner, Feynman Diagrams: Modeling between Physics and Mathematics, Perspectives on Science 26 (2018) 482.
- [9] K.G. Chetyrkin and F.V. Tkachov, Integration by Parts: The Algorithm to Calculate beta Functions in 4 Loops, Nucl. Phys. B192 (1981) 159.
- [10] http://www.feyndiagram.com
- [11] J.A.M. Vermaseren, Axodraw, Comput. Phys. Commun. 83 (1994) 45. https://www. nikhef.nl/~form/maindir/others/axodraw/axodraw.html

- [12] T. Ohl, Drawing Feynman diagrams with Latex and Metafont, Comput. Phys. Commun. 90 (1995) 340, hep-ph/9505351.https://ctan.org/pkg/feynmf
- [13] http://osksn2.hep.sci.osaka-u.ac.jp/~taku/osx/feynmp.html
- [14] J.C. Collins and J.A.M. Vermaseren, Axodraw Version 2, arXiv:1606.01177 [cs.OH].https://ctan.org/pkg/axodraw2
- [15] https://pyfeyn.hepforge.org
- [16] J. Ellis, TikZ-Feynman: Feynman diagrams with TikZ, Comput. Phys. Commun. 210 (2017) 103, arXiv:1601.05437 [hep-ph].
- [17] http://www.luatex.org
- [18] D. Binosi, J. Collins, C. Kaufhold, and L. Theussl, JaxoDraw: A Graphical user interface for drawing Feynman diagrams. Version 2.0 release notes, Comput. Phys. Commun. 180 (2009) 1709, arXiv:0811.4113 [hep-ph].http://jaxodraw.sourceforge. net
- [19] https://feynman.aivazis.com
- [20] https://www.aidansean.com/feynman/
- [21] T. Hahn and P. Lang, FeynEdit: A Tool for drawing Feynman diagrams, Comput. Phys. Commun. 179 (2008) 931, arXiv:0711.1345 [hep-ph].
- [22] http://www.robert-harlander.de/software/aximate